REPUBLIQUE TUNISIENNE
MINISTERE DE L'EDUCATION

INFORMATIQUE

4ème année de l'enseignement secondaire

Sections :
- **Mathématiques**
- **Sciences expérimentales**
- **Sciences techniques**

Les auteurs

Kamel BEN RHOUMA
Maître Assistant à l'ENSI

Rached DOUARI
Inspecteur principal

Slim GHARBI
Professeur de l'enseignement secondaire

Les évaluateurs

Moncef GAFSI
Maître Assistant à l'ENSI

Abdelhafidh ABIDI
Inspecteur principal

Centre National Pédagogique

Préface

Le présent manuel est conforme au nouveau programme d'informatique de 4ème année des sections scientifiques à savoir la section Mathématiques, Sciences Expérimentales et Sciences Techniques. Il est composé de six chapitres répartis en leçons. Chaque leçon est composée d'un nombre de concepts et d'activités homogènes. Nous pensons que cette partition aidera le lecteur à se faire une représentation claire des différents objets d'apprentissage contenus dans ce livre. Notons que plusieurs concepts ont été vus en classe de troisième et par conséquent l'un des objectifs de ce livre est de les rappeler et de les approfondir tout en ajoutant d'autres apprentissages.

Notons que l'objectif fondamental reste la résolution de problèmes, la déduction d'algorithmes et la traduction des ces derniers en programmes informatiques exécutables par un ordinateur.

Sur le plan méthodologique, nous allons continuer d'adopter l'approche inductive qui, grâce à des activités simples et ciblées et d'une façon progressive, conduira l'élève à découvrir par lui même les concepts préconisés. Nous allons continuer cet apprentissage en utilisant la même approche qu'en classe de troisième en l'occurrence l'approche descendante. Nous utiliserons la grille d'analyse qui aide à l'organisation et la systématisation de l'analyse d'un problème. Quant au langage de programmation, nous allons continuer à développer en Pascal pour des raisons de commodités pédagogiques et logistiques.

Sur le plan progression conceptuelle, nous commençons ce livre par les structures de données puis on aborde les différentes structures de contrôle. Les leçons qui composent chaque chapitre traitent des notions proches et cohérentes. Le premier chapitre traite les structures simples, les types standards et les différentes représentations que leur réserve la plupart des langages et particulièrement le langage Pascal. Le second chapitre est réservé aux actions élémentaires simples. On y fait un rappel sur l'affectation, ensuite l'action de sortie et enfin l'action d'entrée. Nous traitons ensuite les structures de contrôle conditionnelles avec ses différentes formes : simple et généralisée. Dans le quatrième chapitre, nous traitons les structures de contrôle itératives complètes et les structures de contrôle itératives à condition d'arrêt. Cela va nous permettre de résoudre entre autres les problèmes récurrents. Ces outils sont à la base du développement et de l'écriture de modules que ce soient fonctions ou procédures. En effet, les sous-programmes sont à la base de l'approche modulaire adoptée dans ce livre. Nous terminons ce livre par les traitements avancés où nous développons les méthodes de tri usuelles à savoir le tri par sélection, le tri à bulles et le tri par insertion ainsi que les méthodes de recherche dans un tableau à savoir la recherche séquentielle et la recherche dichotomique.

Chaque concept est présenté à travers une ou plusieurs activités permettant à l'élève de construire son savoir et de déduire lui-même les notions visées. Une récapitulation concise termine chaque leçon en permettant de retenir les points importants du cours. Nous proposons à la fin de chaque leçon ou chapitre une série d'exercices à difficulté croissante permettant aux lecteurs de s'exercer et de s'auto évaluer. Le quota horaire par leçon n'est pas forcément une séance de deux heures.

Nous mettons à la disposition du lecteur une bibliographie référencielle de ce livre. Elle pourra vous guider pour d'autres lectures.

Nous espérons que ce manuel vous apportera l'aide nécessaire pour réussir un bon apprentissage du programme correspondant. Toutefois, nous vous serons reconnaissants de bien vouloir nous faire part de vos suggestions et de vos remarques.

Les auteurs.

Sommaire

CHAPITRE

Chapitre I

Les structures de données

Objectifs :

- Montrer l'importance de la notion de variable

- Identifier et manipuler les différents types de données

Plan du chapitre :

Nous rappelons dans ce chapitre des notions fondamentales telles que la constante, la variable, les types de données, les différents types d'expression et les tableaux.
Toutes ces entités sont manipulées dans les algorithmes.
Un algorithme est une suite finie d'instructions permettant de résoudre un problème.

Leçon 1

Les variables et les types standard de donnees

Objectifs spécifiques :

- Identifier et manipuler les constantes et les variables.
- Manipuler les types standard.
- Manipuler les procédures et les fonctions prédéfinies.

Plan de la leçon :

Leçon 1

Les variables et les types standard de donnees

C'est une simplification en présense de l'objet concret infiniment complexe et perpétuellement changeant, simplification qui nous est imposée soit par les nécessités de l'action soit par les exigences de l'entendement, et qui consiste à considérer un élément de l'objet comme isolé alors que rien n'est isolable, et comme constant alors que rien n'est en repos.

ALAIN

I. Les constantes et les variables

Définition

Une constante est un objet ayant une valeur fixe tout le long de l'exécution d'un algorithme ou d'un programme.

Une constante est caractérisée par :
- son nom (un identificateur unique).
- sa valeur.

Activité 1

Identifier quelques constantes utilisées dans votre cours de mathématique.

Réponse :
La constante g, utilisée dans le calcul du poids, vaut 9.81. La constante pi, utilisée dans le calcul trigonométrique, vaut 3.141. Ces constantes seront déclarées dans le langage PASCAL de la manière suivante :

```
CONST   g = 9.81 ;
        pi = 3.141 ;
```

Définition

On appelle variable un objet pouvant prendre différentes valeurs tout le long de l'exécution d'un algorithme ou d'un programme.

Une variable est caractérisée par :
- son nom (un identificateur unique)
- son type
- son contenu.

Remarques :

1- Le choix de l'identificateur d'un objet doit être fait de manière à être le plus significatif possible.

2- Lorsqu'on déclare une variable, on réserve en mémoire vive (RAM) un espace mémoire propre à la variable. En effet, la variable constitue le moyen de stocker les données. C'est pour cette raison que la notion de variable est une notion fondamentale en algorithmique. Généralement, l'opération permettant de changer le contenu d'une variable est l'affectation.

Pour réussir un algorithme, il faudra d'une part déclarer les différentes variables utiles et nécessaires pour le bon déroulement de l'algorithme en définissant correctement leurs types et d'autres parts leur donner les valeurs correctes tout le long de l'algorithme. Certaines variables serviront de données initiales, d'autres seront des variables intermédiaires et d'autres variables seront les résultats de l'algorithme. Bien entendu, une variable peut être une combinaison de deux ou trois classes (donnée, intermédiaire ou résultat).

Activité 2

Dans le calcul d'un salaire annuel, on pourra utiliser la constante nb_mois et les variables Salaire_annuel et Salaire_mensuel.

Salaire_annuel = nb_mois x Salaire_mensuel

1) Elaborer le tableau de déclaration des objets intervenant dans cette affectation.
2) Déclarer les différents objets en Pascal.

Réponses :

1)

Tableaux de déclaration des objets

Objet	Type/nature	Rôle
…	…	…
nb_mois	Constante = 12	Nombre de mois de l'année
Salaire_annuel	Réel	Salaire annuel de l'employé
Salaire_mensuel	Réel	Salaire mensuel de l'employé
…	…	…

2)
```
CONST     nb_mois=12 ;
VAR       Salaire_annuel : REAL ;
          Salaire_mensuel : REAL ;
```

Activité 3

Soit la séquence d'affectations suivante :

1) i := 50 ;
2) j := 70 ;
3) k := i ;
4) i := j ;
5) j := k ;

1) Donner le résultat de l'exécution de cette séquence.
2) Quelles sont les valeurs finales de i et de j ?
3) Quel est le rôle de cette séquence ?
4) Quelle est l'utilité de la variable k ?

Réponses :
1) Nous pouvons suivre l'évolution des valeurs des variables i, j et k en les regroupant dans un tableau constituant la trace de la séquence :

Trace de la séquence			
N° de l'instruction	**i**	**j**	**k**
1	50	-	-
2	50	70	-
3	50	70	50
4	70	70	50
5	70	50	50

A l'instruction N°1, seule la variable i prend la valeur 50.
A l'instruction N°2, la variable j prend la valeur 70 mais la variable i est restée inchangée.
A l'instruction N°3, k a pris la valeur de i c'est à dire 50, i et j sont restées inchangées.
A l'instruction N°4, i a changé et a pris la valeur de j c'est à dire 70 et j et k sont restées inchangées.
A l'instruction N°5, j a pris la valeur de k c'est à dire 50, i et k sont restées inchangées.

2) Les valeurs finales de i et de j sont respectivement 70 et 50.

3) Cette séquence a permis la permutation des contenus des deux variables i et j.

4) La variable k a assuré la sauvegarde du contenu de i avant de lui affecter j par l'instruction N°4. Elle est appelée variable intermédiaire, temporaire ou auxiliaire.
En effet, si on affecte à i le contenu de j sans le sauvegarder dans k, il sera perdu.

II. Les types de données

Le type d'une variable permet de déterminer le domaine des valeurs possibles que peut prendre cette variable. La connaissance du type permet également de déduire l'ensemble des opérateurs applicables sur les variables de ce type ainsi que l'espace mémoire en octets qui sera réservé à ces variables.

Un type est désigné par un identificateur (ou un nom). Les types standard sont :

- Le type **Entier**, désignant les valeurs des nombres entiers relatifs.
- Le type **Réel**, désignant les valeurs des nombres réels.
- Le type **Caractère**, désignant les "valeurs" des caractères.
- Le type **Booléen**, désignant les valeurs logiques.
- Le type **Chaîne de caractères**, désignant les "valeurs" des chaînes de caractères.

II. 1 Les types numériques

Dans la catégorie des types numériques, nous pouvons distinguer les types Entier et Réel.

II.1.1 Le type entier

Activité 4

1- Quel est l'ordre de grandeur de la valeur maximale d'un entier ?
2- Rappeler les opérateurs arithmétiques sur les entiers.
3- Evaluer les expressions arithmétiques suivantes :
  a- 17 DIV 5
  b- 17 MOD 5
  c- ((58 DIV 7) MOD 2) + 5
  d- (49 MOD 17) DIV (4 * 3)
4- Est- ce qu'on pourra utiliser le type entier pour représenter les quantités suivantes?
  a- Nombre de jours de l'année
  b- Durée en heures d'une séance de cours ou TP
  c- Nombre de jours du mois de février
  d- Salaire mensuel exprimé en dinars d'un employé.
5- Définir l'effet de débordement pour une variable de type entier.
6- Déclarer trois variables entières i, j et k en Pascal.

Réponses :

1- Sachant que les entiers en mathématiques forment un ensemble infini nommé Z. Or, en informatique, un entier a une représentation en machine limitée à un nombre fini d'octets (généralement 2 octets). La valeur maximale d'un entier est égale à 32767.
2- Les opérateurs arithmétiques sont **+ , - , * , DIV** (Donne le quotient dans la division entière) et **MOD** (Donne le reste de la division entière).

3-a- Le résultat de cette expression est 3
b- Le résultat de cette expression est 2
c- Le résultat de cette expression est 5
d- Le résultat de cette expression est 1

4-a- Le nombre de jours de l'année est un nombre entier (365 ou 366).
b- La durée en heures d'une séance est un nombre réel car la séance peut être par exemple de 1.5 heures.
c- Le nombre de jours du mois de février est un entier.
d- Le salaire mensuel exprimé en dinars d'un employé est un nombre réel.

5- Quand on manipule une variable de type entier, il faut faire attention au risque de débordement. Quand il y a débordement au-delà des valeurs Minimale et Maximale, les calculs deviennent erronés ou provoquent des erreurs d'exécution selon les langages utilisés.

6-
```
VAR i, j, k : INTEGER;
```

Remarque :

Le langage Pascal a cinq types entiers prédéfinis. Chaque type a un domaine de définition spécifique.

Type	Domaine de définition	Nombre de bits
SHORTINT	-128..127	Signé 8 bits
INTEGER	-32768..32767	Signé 16 bits
LONGINT	-2147483648..2147483647	Signé 32 bits
BYTE	0..255	Non signé 8 bits
WORD	0..65535	Non signé 16 bits

II.1.2 Le type réel

Activité 5

1- Quel est le domaine des valeurs du type réel ?
2- Quels sont les opérateurs arithmétiques valides sur des variables de type réel?
3- Déclarer deux variables x et y de type réel en Pascal.

Réponses :
1- Ce type recouvre un sous-ensemble de l'ensemble des nombres réels IR. La définition de ce sous-ensemble est liée à la représentation en binaire des réels dans la machine. Un nombre réel peut occuper 6 octets en Pascal sur certaines machines. On peut le coder ainsi dans l'intervalle de -10^{38} à 10^{38}.

Exemples de nombres réels

0.	-55.36	3.14	60 10^{-9}	1.23	-38.0	5.6 10^{6}

5.6 10^{6} c'est-à-dire 5 600 000 s'écrira aussi 5.6E+6. La lettre E se lit : «dix puissance»

2- Ces opérateurs sont + , - , * et / (division réelle).

3- `VAR x, y : REAL;`

II.1.3 Les fonctions arithmétiques standards

Tous les langages de programmation offrent une bibliothèque de fonctions arithmétiques facilitant la réalisation de certains calculs. Dans l'activité suivante, nous étudions quelques unes d'entre elles.

Activité 6

Soient les fonctions arithmétiques suivantes :

Tronc(-8.224)	Tronc(3.141)	Tronc(334.8)
Arrondi(712.499)	Arrondi(12.50)	Arrondi(12.99)
Abs(-7)		
Carré(7)		
RacineCarré(2)		
Sin(1.5705)		
Cos(1.5705)		
Tang(3.141)		
Cotang(1.5705)		
Ent(3.7)	Ent(-5.5)	
Aléa	Aléa(7)	
Ln(1.0)		
Exp(0.0)		

Elaborer un tableau de 6 colonnes donnant le nom algorithmique de chacune des fonctions précédentes, le code en pascal, le type du paramètre (nombre entre parenthèses), le rôle de chacune des fonctions et les résultats d'évaluation.

Réponse :

Nom algorith-mique	Code en Pascal	Type du paramètre x ou n	Type du Résultat	Rôle	Exemples
Tronc (x)	TRUNC (x)	Entier ou réel	Entier	supprime la partie décimale pour ne laisser que la composante entière de x.	Tronc (-8.224) vaut -8 Tronc (3.14) vaut 3 Tronc (334.8) vaut 334
Arrondi (x)	ROUND (x)	Entier ou réel	Entier	donne un entier qui est la valeur du réel x arrondie à la plus proche valeur.	Arrondi(712.499) vaut 12 Arrondi (12.50) vaut 13 Arrondi (12.99) vaut 13
Abs (x)	ABS (x)	Entier ou réel	Entier ou Réel (même Type que x)	donne la valeur absolue de x.	Abs (-7) vaut 7
Carré (x)	SQR (x)	Entier ou réel	Entier ou Réel (même Type que x)	donne le carré de x.	Carré (7) vaut 49
Racine Carré (x)	SQRT (x)	Entier ou réel	Réel	donne la racine carrée de x si x n'est pas négatif et provoque une erreur, sinon.	RacineCarré (2) vaut 1.414 …
Sin (x)	SIN (x)	Entier ou réel	Réel	donne le sinus de x (x en radians).	Sin(1.5705) vaut 1
Cos (x)	COS (x)	Entier ou réel	Réel	donne le cosinus de x (x en radians).	Cos(1.5705) vaut 0
Tang (x)	TAN (x)	Entier ou réel	Réel	donne la tangente de x. (x en radians).	Tang(3.141) vaut 0
Cotang (x)	COTAN (x)	Entier ou réel	Réel	donne la cotangente de x. (x en radians).	Cotang(1.5705) vaut 0
ENT(x)	INT (x)	Entier ou réel	Entier	donne la partie entière d'un réel.	ENT(3.7) vaut 3 ENT(-5.5) vaut -6
Aléa	RANDOM	Entier	Entier	donne un réel compris entre 0 et 1 exclus.	Aléa pourrait produire 0.36 par exemple.
Aléa(n)	RANDOM (n)			donne un entier entre 0 et n-1	Aléa(7) pourrait produire 2 par exemple.
Ln(x)	Ln(x)	Entier ou réel	Réel	renvoie le logarithme népérien d'un réel x	Ln(1.0) vaut 0
Exp(x)	Exp(x)	Entier ou réel	Réel	renvoie l'exponentiel de x	Exponentiel de (0.0) vaut 1

Activité 7

Soit la séquence d'affectations suivante :
a ← 3
b ← 2.5
masse ← 12
longueur ← 4.5
largeur ← 1.5
hypo ← RacineCarré (carré(a)+carré(b))
poids ← masse * g
surface ← longueur * largeur

1- Déclarer en Pascal les différents objets.
2- Traduire en Pascal les différentes affectations.

Réponses :

1-
```
CONST g=9.81;
VAR   masse : INTEGER;
      a,b,poids,longueur,largeur,surface,hypo : REAL;
```

2-
```
a:=3;
b :=2.5 ;
masse :=12 ;
longueur :=4.5 ;
larguer :=1.5 ;
hypo := SQRT(SQR(a)+SQR(b));
poids := masse * g ;
surface :=  longueur * largeur ;
```

II.2 Le type booléen

Le type booléen est utilisé pour caractériser des objets de type logique.

Activité 8

1- Quelles sont les valeurs du type booléen?
2- Quels sont les opérateurs logiques qu'on peut appliquer sur les booléens?
3- Evaluer les propositions logiques suivantes :
a- La proposition (88>66) ET (44<55)
b- La proposition (88>66) ET (66<55)

Réponses :

1- Les variables d'un tel type peuvent prendre uniquement deux valeurs logiques **VRAI** et **FAUX** (**TRUE** ET **FALSE** en **Pascal**).

2- Ces opérateurs sont NON (négation), Et (conjonction), OU (disjonction) et OUex (OU exclusif).

3- a- (88>66) ET (44<55)

VRAI ET VRAI

VRAI

3- b- (88>66) ET (66<55)

VRAI ET FAUX

FAUX

Activité 9

1- Soient a, b et c trois variables booléennes
a- Donner les étapes d'évaluation de l'expression a OU b ET c.
b- Si on veut d'abord évaluer la disjonction a OU b, comment faut-il s'y prendre ?

2- Dans la réponse à la question précédente, nous avons utilisé le fait que l'opérateur ET est prioritaire par rapport à l'opérateur OU. Donner l'ordre de priorité qui existe entre les opérateurs logiques.

3- Déclarer en Pascal quatre variables p, q, existe et drapeau de type booléen.

4- Evaluer les expressions logiques suivantes :
a- (x <= 4) ET (x >= 1) avec x = 3
b- (x <= 2) ET (x >= 0) avec x = -4
c- NON (x <= 55) OU (x >= 0) avec x = 21

Réponses :

1- a- on évalue d'abord la conjonction b ET c puis la disjonction a OU (b ET c) comme s'il y avait des parenthèses.
b- il suffit d'ajouter des parenthèses (a OU b) ET c.

2- Il existe un ordre de priorité entre les opérateurs logiques :
- La négation NON est prioritaire par rapport à la conjonction ET.
- La conjonction ET est prioritaire par rapport à la disjonction OU.
- La disjonction OU a la même priorité que l'opérateur OUex.

Si deux opérateurs ont la même priorité, le calcul de l'expression logique se fera de gauche à droite.
Dans tous les cas, les opérations mises entre parenthèses sont prioritaires.
Les parenthèses les plus internes sont les plus prioritaires.

3-
```
VAR p, q, existe, drapeau : BOOLEAN ;
```

4- a- VRAI
b- FAUX
c- VRAI

II.3 Le type caractère

Le type caractère est utilisé pour représenter une lettre minuscule, une lettre majuscule, un chiffre ou un signe de ponctuation, mais aussi un espace typographique, une tabulation, un retour à la ligne et quelques autres opérations spéciales (sonnerie, effacement, etc.). Tous les caractères sont ordonnés selon leur code ASCII (voir Annexe à la fin du livre).

Activité 10

1- Les chiffres, les lettres en majuscule, les lettres en minuscule ont des codes ordonnés et contigus ; Quel est cet ordre ?
2- Citer quelques opérateurs applicables aux caractères.
3- Déclarer en Pascal :
a- une constante caractère de valeur "v"
b- deux variables de type caractère car1 et car2

Réponses :

1- On peut comparer les caractères comme suit :
"0" < "1" … <"9" < … "A" < "B" … < "Z" … "a" < "b" <"z" …

2- Comme les valeurs de type caractère sont ordonnées, les opérateurs relationnels y sont définis.
Exemples :
"B" < "F" est une proposition VRAIE.
"g" > "b" est une proposition VRAIE.

3- **a-**
```
CONST  car  = 'v' ;
```
b-
```
VAR  car1 , car2 : CHAR ;".
```

Les fonctions prédéfinies

Parmi les fonctions prédéfinies à grand usage, nous citons :
ORD, CHR, SUCC et PRED.

Activité 11

1- Soient les exemples de fonctions prédéfinies suivantes :

ORD ("A")	ORD("a")
CHR (65)	CHR (97)
SUCC ("F")	SUCC ("5")
PRED ("Z")	PRED ("9")
MAJUS("c ")	

En utilisant le tableau ASCII de l'annexe du livre, élaborer un tableau montrant le nom de la fonction, son code en Pascal, son rôle et sa valeur.

2- Evaluer les expressions suivantes :
a- CHR (ORD (c)), c est une variable de type caractère.
b- ORD (CHR (n)), n est un entier.

3- Que désigne chacune des notations suivantes ?
a- t
b- "t"
c- 7
d- "7"

Réponses :

1- Supposons que c est une variable de type caractère et que n est un entier compris entre 0 et 255.

Nom	Code en Pascal	Rôle	Exemples
ORD (c)	**ORD (c)**	renvoie le code ASCII du caractère c. Le résultat est un entier positif.	ORD ("A") vaut 65 ORD ("a") vaut 97
CHR (n)	**CHR (n)**	renvoie le caractère dont le code ASCII est n.	CHR (65) vaut "A" CHR (97) vaut "a"
SUCC (c)	**SUCC (c))**	renvoie le caractère successeur de c s'il existe.	SUCC ("F") vaut "G" SUCC ("5") vaut "6"
PRED (c)	**PRED (c)**	renvoie le caractère prédécesseur de c s'il existe.	PRED ("Z") vaut "Y" PRED ("9") vaut "8'"
MAJUS (c)	**UPCASE(c)**	Convertit le caractère c en majuscule s'il est possible.	MAJUS("c ") vaut "C"

2- **a-** c
b- n

3- **a-** t ainsi écrit, représente un objet identifié par l'identificateur t ; c'est par exemple une variable.
b- t entre guillemets désigne le caractère t.
c- 7 désigne l'entier 7.
d- "7" désigne le caractère 7.

II.4 Le type chaîne de caractères

Une chaîne de caractères est une suite ordonnée de caractères. En algorithmique, la valeur d'une chaîne se note en utilisant des guillemets ("). En langage Pascal, on utilise plutôt des quotes simples (').

Exemple

Algorithmique	Pascal
"baccalauréat"	'baccalauréat'
"l'algorithmique"	'l''algorithmique'

Les variables chaînes de caractères sont définies par une déclaration pouvant indiquer le nombre maximum de ses caractères.

Activité 12

1- Elaborer un tableau de déclaration des objets où vous définirez une chaîne prenom de 10 caractères, une chaîne nom pouvant contenir jusqu'à 255 caractères et une chaîne adresse de 25 caractères.
2- Déclarer ces objets en Pascal.
3- Réaliser les affectations suivantes :
Chaîne vide à la variable Prenom, "Aloui" à la variable Nom et un espace à la variable Adresse.
4- Comment accéder au i ème caractère d'une chaîne CH ?

5- Soit l'affectation suivante :
Nom ← "Beldi"
a- Quelles sont les valeurs de Nom[1] et de Nom[5] ?
b- Après l'affectation Nom[2] ← "a" , Quel est le contenu de la variable Nom.

Réponses :

1- Tableau de déclaration des objets

Objet	Type/Nature	Rôle
Prénom	Chaîne [10]	Prénom d'un individu sur 10 caractères maximum
Nom	Chaîne	Chaîne pouvant contenir jusqu'à 255 caractères.
Adresse	Chaîne [25]	Chaîne pouvant contenir jusqu'à 25 caractères.

2- Dans le langage PASCAL les déclarations correspondantes seront :

```
VAR
      Prenom : String [10] ;
      Nom : String ;
      adresse : String [25] ;
```

Remarque :
À la variable Prenom, on peut affecter une chaîne de caractères ayant au maximum dix caractères. Par contre, il n' y a pas de limitation visible pour la longueur de la variable Nom (Ce nombre maximal est de 255 caractères pour certains compilateurs).

3- Prenom ← "" {vide}
Nom ← "Aloui"
Adresse ← " " {un espace}

4- On pourra accéder en lecture et en écriture au ième caractère d'une chaîne CH en utilisant la notation CH[i] où $1 \leq i \leq$ Long (CH) où Long(CH) désigne la longueur de la chaîne CH.

5- **a-** Nom [1] donne "B"
Nom [5] donne "i"
b- Nom devient "Baldi"

Les fonctions et les procédures prédéfinies

Les langages de programmation offrent un jeu de fonctions et de procédure prédéfinies. Nous étudions dans la suite les fonctions et les procédures les plus usuelles.

Activité 13

1- Soient les exemples de fonctions et procédures prédéfinies suivantes :

concat ("Bon","jour")
Sous_chaîne ("Baccalauréat",6,7)
Convch (2008,ch)
Efface (v, 7,3) où v contient "Disquette"
Insère ("tte",v,7) où v contient "Disque"
Long ("Lycée")
Pos ("Bac","Baccalauréat")
Valeur ("16.5",d,erreur)

Elaborer un tableau de 4 colonnes donnant le nom algorithmique de chacune des procédures ou fonctions précédentes, le code en pascal, le rôle de chacune des procédures ou fonctions et les résultats générés par les exemples définis ci-dessus.

Algorithmique	Code en Pascal	Rôle	Exemples
Concat(ch1,ch2, ..,chN)	**CONCAT (ch1, ch2, … chN)**	Concat retourne la concaténation des chaînes ch1, ch2, ..., chN	concat ("Bon","jour") retourne la chaîne "Bonjour"
Long (ch)	**LENGTH (ch)**	Long retourne un entier représentant la longueur en caractères de la chaîne ch. Ce nombre se trouve aussi dans le caractère ch[0].	Long ("Lycée") retourne l'entier 5
Sous_chaîne (ch, p, nbc)	**COPY (ch, p, nbc)**	Fonction qui retourne une sous-chaîne d'une longueur **nbc** à partir de la position **p** dans **ch**.	Sous_chaîne("Baccalauréat",6,7) retourne la chaîne "lauréat"

Pos (ch1, ch2)	**POS (ch1, ch2)**	Pos retourne la première position de la chaîne ch1 dans la chaîne ch2	Pos("Bac","Baccalauréat") retourne l'entier 1
Efface (ch, p, n)	**DELETE (ch, p, n)**	Efface est une procédure qui enlève **n** caractères de **ch** à partir de la position **p**.	Efface(v, 7,3) modifie v qui contiendra "Disque"
Insère(ch1,ch2, p)	**INSERT(ch1, ch2, p)**	Procédure qui insère la chaîne **ch1** dans la chaîne **ch2** à partir de la position **p**. Le caractère numéro **p** et les suivants sont décalés vers la droite.	Insère ("tte",v,7) modifie v qui contiendra "Disquette"
Convch (d, ch1)	**STR (d, ch1)**	Procédure qui convertit un nombre décimal d en chaîne de caractères et l'affecte à la variable **ch1**.	Convch(2008, ch) modifie ch qui contiendra "2008"
Valeur (ch, d, erreur)	**VAL (ch, d, erreur)**	Procédure qui convertit une chaîne ch en une valeur numérique décimale et l'affecte à la variable d. Le paramètre erreur est une variable de type entier qui contiendra 0 si la conversion s'est bien déroulée, sinon elle contiendra la position du caractère qui a déclenché l'erreur.	Valeur("16.5",d,erreur) modifie d qui contiendra le réel 16.5. erreur contiendra 0.

Retenons

• Les constantes contiennent des valeurs inchangées tout le long de l'exécution de l'algorithme. Les variables peuvent changer de contenu tout le long de l'algorithme.

• Un type définit un ensemble de valeurs et un ensemble d'opérations applicables sur ces valeurs.

• Les types standard sont :
  – les entiers,
  – les réels,
  – les booléens,
  – les caractères,
  – et les chaînes de caractères

Exercices

Exercice 1

Quelle est la différence entre une constante et une variable?

Exercice 2

Évaluer le contenu des variables m, n, p suite à l'exécution de chacune des séquences suivantes :

Séquence 1	Séquence 2
1) m ← 30 2) n ← m+ 20 3) p ← n +m 4) m ← n - p	1) m ← 2 2) m ← m*m 3) m ← m *m 4) m ← m*m

Exercice 3

1- Évaluer le contenu des variables a, b, et c suite à l'exécution de chacune des séquences suivantes :

Séquence 1	Séquence 2	Séquence 3
1) a ← 3 2) b ← 5 3) a ← b 4) b ← a	1) a ← 3 2) b ← 5 3) c ← b 4) b ← a 5) a ← c	1) a ← 7 2) b ← 3 3) b ← a + b 4) a ← b - a 5) b ← b -a

2- Quel est le rôle de la variable c dans la séquence 2 ?

Exercice 4

On se propose de calculer la surface d'un rectangle. Etablir un tableau de déclaration des objets nécessaires.

Exercice 5

Dire si l'utilisation de ces identificateurs est permise dans le langage PASCAL ou non. Justifier votre réponse.

- Code Produit
- Code+Produit
- Code_Produit
- 3etapes
- capacité

Exercice 6

Le programme Pascal suivant comporte de nombreuses erreurs, trouver-les.

```
PROGRAM  deserreurs ;
     CONST
          I=10 ; J=40 ; K=5 ;
     TYPE
        integer = -32000..+32000
        voyelles1=('A','E','I','O','U');
        voyelles2=(A,E,I,O,U);
        Abscisses=0..0.001;
        Indice= -10..+10;
     VAR
          S:Indice ; V: voyelle2 ; R: REAL;
     BEGIN
         R:=35;
         V:=R+1;
         S:=2*J+K;
     END
```

Exercice 7

Dire si l'on pourra utiliser le type entier pour représenter les quantités suivantes :

a- La note obtenue dans un examen
b- Le nombre de matières étudiées pendant un trimestre
c- Le coefficient d'une matière
d- La moyenne générale du trimestre

Exercice 8

Ecrire les instructions PASCAL permettant de réaliser les objectifs suivants :

1- Obtenir la longueur de la chaîne "Informatique"
2- Récupérer la sous-chaîne de 5 caractères à partir de la position 8 de la chaîne "Informatique"
3- Effacer les 5 premiers caractères de la chaîne "Informatique"

Exercice 9

a est un entier, x est un réel et ch est un caractère.

Dire pourquoi les instructions suivantes sont erronées, et les corriger dans la mesure du possible.

- a:=3.4
- x:=5,16
- ch:=x
- a:=maxent+5

Exercice 10

Soit la partie déclarative du programme Pascal suivant :

```
PROGRAM  Dates;
      CONST
         Date1="03/07/2006";
      VAR
         Date2: STRING[10];
         Jour: STRING[2];
         Mois: STRING[2];
         Annee: STRING[4];
        ......
```

Compléter le corps du programme par les instructions (bloc BEGIN...END) permettant de réaliser les traitements suivants :

- Mettre Date1 dans Date2 sous la forme jjmmaaaa (indication: Date2 va contenir 03072006)
- Mettre le jour dans la variable Jour, le mois dans la variable Mois et l'année dans la variables Annee.

Exercice 11

Soit la partie déclarative du programme Pascal suivant :

```
PROGRAM  mots ;
  CONST
      m1: “abcd”;
      m2: “ordinateur”;
      m3: “scalaire”;
  VAR
    symetrie: STRING[8];
    compose: STRING ;
        ......
```

Compléter la partie du programme Pascal précédente en ajoutant des variables (si c'est nécessaire) et le bloc BEGIN … END pour :

- Avoir dans la variable « symetrie » une chaîne de caractères symétrique obtenue à partir de la variable « m1 ».
- Avoir dans la variable « compose » une chaîne de caractères obtenue en combinant la première moitié de « m2 » et la deuxième moitié de « m3 » en utilisant les fonctions : concat() et copy().

Exercice 12

Ecrire un programme Pascal qui permet de transformer la chaîne de caractères “informatique” en majuscule sans utiliser la fonction upcase(c), et la mettre dans une variable nommée maj.

Indication : le code ASCII de 'a' est 97 et le code ASCII de 'A' est 65.

Leçon 2

Les expressions

Objectifs spécifiques :

- Reconnaître les expressions
- Evaluer des expressions en tenant compte de la priorité des opérateurs
- Utiliser des fonctions prédéfinies dans des expressions

Plan de la leçon :

Leçon 2

Les expressions

J'appelle claire (la connaissance) qui est présente et manifeste à un esprit attentif : de même que nous disons voir clairement les objets lorsque étant présents ils agissent assez fort, et que nos yeux sont disposés à les regarder.

DESCARTES

Une expression simple est constituée d'opérandes reliés par des opérateurs.
Nous distinguons deux types d'expressions :

- les expressions arithmétiques donnant une valeur numérique.
- les expressions logiques donnant une valeur booléenne.

Activité 1

Donner le type et la valeur de chacune des expressions suivantes.
1- (6 < = 2) **2-** (7+3) DIV 2

Réponses :
1- C'est une expression logique ; le résultat est booléen ; sa valeur est FAUX.
2- C'est une expression arithmétique ; le résultat est entier ; sa valeur est 5.

I. Les opérandes

Activité 2

1- Soient les expressions suivantes :
a- 5 + 3.141 **b-** 7 MOD 3
Indiquer pour chacune des expressions ci-dessus le premier et le deuxième opérande et l'opérateur.
2- Quels sont les types d'opérandes ?
3- Donner les valeurs résultats de chacune de ces expressions :
a- 3.14* D1 **b-** (3.14*D1) + (3.14 *D2)
c- 3.14 * CARRE (R1) **d-** 100/25
pour les valeurs D1=4, D2=3 et R1=2

Réponses :
1- a- 5 est le premier opérande ; + est l'opérateur et 3.141 constitue le deuxième opérande.
b- 7 est le premier opérande ; MOD est l'opérateur et 3 constitue le deuxième opérande.
2- Les opérandes peuvent être des constantes, des variables, des valeurs ou des résultats envoyés par des fonctions. Par ailleurs, un opérande peut être une expression.
3- a- 12.56 **b-** 21.98 **c-** 12.56 **d-** 4.0

II. Les opérateurs

II.1 Les opérateurs arithmétiques

Les opérateurs unaires

Un opérateur est dit unaire s'il est appliqué à un seul opérande. On dit aussi qu'il est monadique.

L'opérateur unaire usuel

Exemple : - (77) **-** est l'opérateur et **77** est l'opérande.

Les opérateurs binaires

Un opérateur est dit binaire s'il est appliqué à deux opérandes. On dit aussi qu'il est dyadique.

Opérateur	Type opérande	Type résultat
–	Entier ou Réel	Entier ou Réel

Activité 3

1- Remplir un tableau permettant de montrer le type du résultat pour les différents types d'opérandes admis en ce qui concerne les opérateurs binaires multiplicatifs.

2- Remplir un tableau permettant de montrer le type du résultat pour les différents types d'opérandes admis en ce qui concerne les opérateurs binaires additifs.

Réponses :

1-

Opérateur	type opérande 1	type opérande 2	type résultat
*	Entier	Entier	Entier
	Réel	Réel	Réel
	Réel	Entier	Réel
	Entier	Réel	Réel
/	Entier	Entier	Réel
	Entier	Réel	Réel
	Réel	Entier	Réel
	Réel	Réel	Réel
DIV	Entier	Entier	Entier
MOD	Entier	Entier	Entier

2-

Opérateur	type opérande 1	type opérande 2	type résultat
+,-	Entier	Entier	Entier
	Entier	Réel	Réel
	Réel	Entier	Réel
	Réel	Réel	Réel

II.2 Les opérateurs logiques

Activité 4

1- a- Quel est l'opérateur unaire usuel, le type de l'opérande et le type du résultat?
b- Evaluer l'expression suivante : NON (44<66)

2- Remplir un tableau permettant de montrer le type du résultat pour les types d'opérandes admis en ce qui concerne les opérateurs binaires logiques.

Réponses :

1- a-

Opérateur	Type opérande	Type résultat
NON	Booléen	Booléen

b- NON est l'opérateur et (44<66) est un opérande. Le résultat de cette expression est FAUX.

2-

Opérateurs	type opérande 1	type opérande 2	type résultat
ET	Booléen	Booléen	Booléen
OU	Booléen	Booléen	Booléen
Ouex	Booléen	Booléen	Booléen

NB : Souvent les opérateurs logiques sont notés multiplicativement pour le ET et additivement pour le OU.

II.3 Les opérateurs relationnels

Le tableau suivant illustre les opérateurs relationnels usuels :

Opérateurs	Code en Pascal	Type opérande 1	Type opérande 2	Type résultat
<, >, =, <>, <=, ≥	<, >, = , <>, <=, >=	Tout type ordonné	Tout type ordonné	Logique

Remarque :
Tous les types que nous avons vus sont des types ordonnés. Toute comparaison entre deux éléments de même type ou de types compatibles est possible.

Activité 5

1- Quel est le résultat d'une comparaison de deux éléments de même type ou de types compatibles ?

2- Pour x=32 et y=36 , quelle est la valeur de x<y ?

Réponses :

1- Le résultat de la comparaison est booléen (VRAI ou FAUX).

2- x < y vaut VRAI

III. Évaluation d'une expression

Lors de l'évaluation d'une expression, on tient compte de la priorité entre les opérateurs.

Activité 6

1- Quel est l'ordre de priorité dans le calcul des expressions ?
2- Donner les étapes de calcul de l'expression 31+7*10.
3- Evaluer l'expression 55 + 6 - 10
4- Dans le cas où on veut imposer un autre ordre, que doit-on faire ?
5- Evaluer les expressions suivantes :
a- (44 * x + y) avec x = 2 et y =4
b- (y + 44 * x) avec x = 2 et y=4
c- ((y + 44) * x) avec x = 2 et y =4

Réponses :
1- Cet ordre est le suivant :
1) Les parenthèses.
2) Les opérateurs unaires.
3) Les opérateurs multiplicatifs.
4) Les opérateurs additifs.
5) Les opérateurs relationnels.
2- 1ère opération 7 * 10 = 70 2ème opération 31 + 70 = 101
3- Le calcul de 55 + 6 -10 commence par 55 + 6 donc 61 puis 61-10 pour avoir 51. En effet, pour les opérateurs de même priorité, on commence par celui qui est le plus à gauche.
4- Dans ce cas, on doit utiliser des parenthèses.
31 + 7 * 10 vaut 101 mais si on écrit (31 + 7) * 10 le résultat est 380.
5- a- 92 **b-** 92 **c-** 96

Retenons

Une expression est composée d'opérateurs et d'opérandes et son évaluation produit une valeur.
- On distingue les expressions arithmétiques, et les expressions logiques.
- L'évaluation d'une expression se fait toujours selon l'ordre de priorité des opérateurs.

Exercices

Exercice 1

1- Donner les expressions arithmétiques correspondantes aux expressions suivantes écrites en Pascal :

sqrt(2 * a + 3 / b - 2) / 3 + x

4 * x / c * 6 - x

10 * x / 2 + 4

2- Réciproquement, écrire en Pascal les expressions arithmétiques suivantes :

$$\frac{4x^2+2x-5}{\sqrt{x+\frac{5y}{2}}} \qquad \frac{y+2}{10x}+1 \qquad \sqrt{\frac{x^2-y}{y-\frac{x}{2}}}$$

Exercice 2

Evaluer les expressions logiques en A, B, C, et D pour chacune des combinaisons (p,q,r,s) suivantes :

1) Pour (p,q,r,s) = (-3, 5, 4, 9)
2) Pour (p,q,r,s) = (3, 7, 4, 9)
3) Pour (p,q,r,s) = (5, 13, 7, 3)

A. (p < q) OU (r > s)
B. (p < q) ET NON (r > s)
C. (p > q) OU (r ≠ p)
D. (p + q < r) ET (p + q > r)

Exercice 3

Cette écriture permettant de vérifier si le caractère C est une voyelle est erronée . Pourquoi ? Qu'aurait-il fallu écrire ?

C='A' **OR** C='E' **OR** C='I' **OR** C='O' **OR** C='U'

Exercice 4

- Rappeler l'ordre de priorités dans lequel une expression doit être évaluée.
- Comment évaluer deux opérateurs ayant la même priorité ?
- Compléter le tableau suivant par le type du résultat :

Opérateur	Type opérande 1	Type opérande 2	Type du résultat
DIV	entier	entier	.....
/	entier	entier	.....
+	réel	entier	.....
-	entier	entier	.....

Leçon 3

Le type scalaire enumere et le type intervalle

Objectifs spécifiques :

- Comprendre l'utilisation du type énuméré.
- Comprendre l'utilisation du type intervalle.
- Manipuler des variables faisant appel aux types énuméré et intervalle.

Plan de la leçon :

Leçon 3

Le type scalaire enumere et le type intervalle

La consience des representations qui suffit pour différencier un objet d'un autre. C'est la clarté. Mais celle qui rend claire la composition des représentations, c'est la distinction.

KANT

Outre les types standard présentés dans la première leçon, nous pouvons définir de nouveaux types appelés souvent types utilisateur.

I. Le type scalaire énuméré

Dans certaines situations où l'on aimerait contraindre une variable à décrire un jeu de valeurs bien déterminé, on utilisera un type scalaire énuméré afin d'énumérer ces valeurs.
Un type scalaire définit, en général, un domaine de valeurs comme c'est le cas du type entier. Par ailleurs, le type scalaire par énumération définit un ensemble ordonné et fini de valeurs désignées par des identificateurs.

Définition

Le type scalaire par énumération définit un ensemble ordonné et fini de valeurs désignées par des identificateurs

Activité 1

Nous voulons définir un type que nous appelons ANNEE_SCOLAIRE contenant les dix mois de l'année scolaire et manipuler les valeurs de ce type.

1- Donner le tableau de déclaration des nouveaux types illustrant le type ANNEE_SCOLAIRE.
2- L'affectation suivante est-elle correcte sachant que mois est une variable de type ANNEE_SCOLAIRE ?

mois ← octobre

3- Quel est l'ordre de ces constantes ?
4- Quels sont les opérateurs applicables à ces valeurs ?
5- Evaluer les expressions suivantes :
a- SUCC (septembre)
b- PRED (juin)
6- La déclaration du type énuméré IMPAIR = (1, 3, 5, 7) est-elle possible ?
7- Déclarer en Pascal
a- un type scalaire énuméré contenant les mois à 30 jours.
b- une variable intitulée mois_court du type mois_a_trente.

Réponses :

1- **Tableau de déclaration des nouveaux types**

Types
ANNEE_SCOLAIRE=(septembre, octobre, novembre, decembre, janvier, fevrier, mars, avril, mai, juin)

septembre, octobre, novembre, decembre, janvier, fevrier, mars, avril, mai, juin sont les éléments du type ANNEE_SCOLAIRE.

2- Une variable mois de type ANNEE_SCOLAIRE peut prendre comme valeur : septembre, octobre, novembre, decembre, janvier, fevrier, mars, avril, mai, juin. L'affectation mois ← octobre est correcte.

3- L'ordre sur ces valeurs est le suivant : septembre < octobre < … < juin. Par ailleurs, on peut appliquer la fonction Ord sur ces valeurs pour déterminer leurs numéros d'ordre, ainsi Ord (septembre) vaut 0 et Ord (octobre) vaut 1 et ainsi de suite.

4- Les opérateurs applicables à ces valeurs sont :
- Les opérateurs de relation
- Les opérateurs **PRED** et **SUCC. PRED** représente le prédécesseur (le précédent) et SUCC représente le successeur (le suivant).

5- **a-** octobre
b- mai

Remarque : **PRED** (septembre) n'existe pas et **SUCC** (juin) n'existe pas non plus et peuvent provoquer des erreurs.

6- Cette déclaration est interdite car 1, 3, 5 et 7 sont des valeurs qui appartiennent au type prédéfini ENTIER.

7- **a-**
```
TYPE mois_a_trente = (avril, juin, septembre, novembre) ;
```
b-
```
VAR  mois_court  : mois_a_trente ;
```

Dans le langage Pascal, la déclaration d'un type scalaire énuméré et celle d'une variable de ce type se font comme suit :

```
TYPE <nom_type>=(constante_1, constante_2, … ,constante_n);
VAR  <nom_variable> : nom_type ;
```

II. Le type intervalle

Le type intervalle possède les propriétés d'un type scalaire discret ordonné (entier, caractère et scalaire énuméré).
La définition d'un intervalle est décrite par la donnée de deux constantes représentant respectivement la "Borne Inférieure" et la "Borne Supérieure" appartenant à un type scalaire discret ordonné et telle que Borne Inférieure < Borne Supérieure.

Activité 2

1- Elaborer un tableau de déclaration des nouveaux types illustrant un type intervalle intitulé mois de 1 à 12 et un type intervalle concernant le premier trimestre de l'année scolaire.

2- a- Quels sont les bornes inférieure et supérieure du type mois ?
b- Quels sont les valeurs que peut prendre une variable du type mois ?
c- Quels sont les bornes inférieure et supérieure du type premier_trimestre ?
d- Quels sont les valeurs que peut prendre une variable du type premier_trimestre?

3- Déclarer en Pascal un type intervalle mois, un type intervalle jours et deux variables mois_actuel et j de types respectifs mois et jours.

Réponses :

1- **Tableau de déclaration des nouveaux types**

Types
Mois = 1. . 12 ANNEE_SCOLAIRE = (septembre, octobre, novembre, decembre, janvier, fevrier, mars, avril,mai, juin) Premier_trimestre = septembre .. decembre

2- a- Les bornes de Mois sont 1 et 12 prises comme valeurs entières.
b- Une variable de type Mois peut prendre ses valeurs entre 1 et 12.
c- Les bornes de premier_trimestre sont septembre et decembre.
d- Une variable de type premier_trimestre peut prendre comme valeur : septembre, octobre, novembre et decembre.

3-
```
TYPE  mois = 1 .. 12;
      jours = 1 .. 31 ;
 VAR  mois_actuel  : mois ;
      j : jours ;
```

En effet, dans le langage Pascal, la déclaration d'un type intervalle et celle d'une variable de ce type se font comme suit :

```
TYPE  <nom_type> = borne_inf .. borne_sup ;
VAR   <nom_variable> : nom_type ;
```

Remarques :

Une variable d'un type intervalle possède toutes les propriétés du type de base dont l'intervalle est issu. Toutefois, sa valeur doit être comprise au sens large entre les bornes de l'intervalle.

L'intérêt de ce type réside dans le fait qu'il permet une meilleure lisibilité de l'algorithme et du programme.

Exemple : mois : 1 . . 12 est beaucoup plus précis que mois : entier

Retenons

Le type scalaire par énumération définit un ensemble ordonné et fini de valeurs désignées par des identificateurs définis par l'utilisateur.

Les opérateurs applicables à ces valeurs sont :

- Les opérateurs de relation
- Les opérateurs PRED et SUCC.

Le type intervalle possède les propriétés d'un type scalaire discret ordonné (entier, caractère et scalaire énuméré).
La définition d'un intervalle est décrite par la donnée de deux constantes représentant respectivement la "Borne Inférieure" et la "Borne Supérieure" appartenant à un type scalaire discret ordonné et telle que Borne Inférieure < Borne Supérieure.

Exercices

Exercice 1

Soit le programme Pascal suivant :

```
PROGRAM erreurs
      TYPE
         eleves  : (Ali,Safa,Sami,Wissem,Kamel) ;
         moyenne = 0..20
      VAR
         e1: eleves;
         e2: eleves;
         n1: moyenne
         n2 : moyenne ;
         reussir : BOOLEAN ;

      BEGIN
         e1 := “Safa”;
         n1  = ORD(Kamel)+2 * SUCC(Ali)  ;
         e2 := Sami  ;
         n2 := -15;
         russir := ( n2 > 10 )  ;
    END.
```

1) Corriger les erreurs du programme Pascal ci-dessus.
2) Evaluer le contenu des variables utilisées dans le Programme.

Exercice 2

Est-ce que la déclaration de l'énumération suivante est correcte? Justifier votre réponse.

Pair = (0,2,4) ;

Exercice 3

En utilisant le type intervalle, déclarer en algorithmique et en Pascal les variables suivantes :

– jour
– mois

Exercice 4

En utilisant le type scalaire énuméré, déclarer en algorithmique puis en Pascal les types suivants :

– couleur_de_base
– jour_de_la semaine

Leçon 4

Les tableaux à une dimension

Objectifs spécifiques

- Comprendre l'utilisation du type tableau.
- Manipuler des tableaux.

Plan de la leçon

Leçon 4

Les tableaux à une dimension

Si, dans la représentation, on laisse de côté les déterminations d'un objet, c'est ce q'on appelle abstraire. Il ne reste alors qu'un objet moins déterminé. C'est à dire un objet abstrait. Mais, si dans la représentation je ne considère q'une détermination singulière de cet ordre, c'est là aussi une représentation abstraite.

HEGEL

Quand on veut regrouper un certain nombre de variables de même type sous un même nom, on utilise la notion de tableau. Un tableau peut être considéré comme une suite de variables, de même nom, repérées par des indices.

Définition :

Un tableau est une structure de données homogènes regroupant un ensemble d'éléments de même type.

I. Déclaration d'un tableau

Dans le cas général, pour déclarer un tableau, on utilisera la forme suivante :

Au niveau de l'analyse et de l'algorithme

Tableau de déclaration des objets

Objet	Type nature	Rôle
Ident _tableau	Tableau de Taille et de Type_élément	

En Pascal

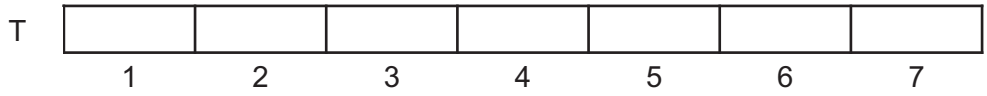
```
VAR
 ident_tableau:ARRAY[Borne_inf..Borne_sup]OF Type_élément;
```

Où :
Ident_tableau : Identificateur du nouveau tableau que nous voulons définir.
Borne_Inf .. Borne_Sup : intervalle correspondant à l'ensemble des valeurs des indices du tableau.

Borne_Inf : Borne inférieure de l'intervalle des indices.
Borne_Sup : borne supérieure de l'intervalle des indices.
Type_élément : type des éléments du tableau. Il peut être l'un des types vu précédemment.

Exemple :

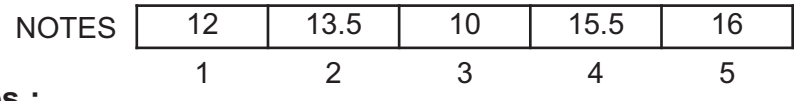
```
T : ARRAY [ 1..7 ] OF REAL ; {déclaration d'un tableau de 7 éléments réels }
```



Suite à cette déclaration, nous réservons un espace mémoire au niveau de la RAM pouvant contenir sept réels.
Un tableau est caractérisé par ses dimensions. Nous nous limiterons à l'étude **des tableaux unidimensionnels** appelés aussi vecteurs.

Activité 1

1- Donner le tableau de déclaration des objets pour un vecteur de 5 éléments de type réel pouvant contenir les notes des élèves.
2- Déclarer le tableau NOTES en Pascal.
3- Affecter à chaque élément la note correspondante suivant le tableau suivant :



Réponses :

1- La déclaration du tableau se fera comme suit :

Tableau de déclaration des objets

Objet	Type nature	Rôle
NOTES	Tableau de 5 réels	Tableau servant à contenir les notes de 5 élèves

2- Dans le langage Pascal, on aurait effectué la déclaration suivante :

```
VAR
   NOTES : ARRAY[1..5] OF REAL;
```

3- NOTES [1] est le 1er élément du tableau NOTES.
NOTES [2] est le 2ème élément du tableau NOTES.
NOTES [3] est le 3ème élément du tableau NOTES.
NOTES [4] est le 4ème élément du tableau NOTES.
NOTES [5] est le 5ème élément du tableau NOTES.

NOTES [1] ← 12
NOTES [2] ← 13.5
NOTES [3] ← 10
NOTES [4] ← 15.5
NOTES [5] ← 16

Remarques :

1) Pour accéder au ième élément du tableau, il suffit de donner l'identificateur du tableau et l'indice i indiquant le rang de l'élément. Cet indice doit être dans l'intervalle Borne_inf.. Borne_sup.

2) Nous pouvons lire et écrire un élément du tableau (voir chapitre 2). Nous pouvons modifier un élément par une instruction d'affectation.
Exemple : NOTES [4] ← 17.50

3) Les opérations possibles sur un élément du tableau sont les mêmes que celles définies sur une variable de même type.

II. Le type tableau

Il est possible de déclarer le type d'un tableau.

Exemple

Tableau de déclaration des nouveaux types

Types
Eleves = tableau de 30 chaînes de caractères
Moyennes = tableau de 30 réels
Comptes = tableau de 26 entiers

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
T_ELEVES	Eleves	Tableau des 30 noms d'élèves
T_MOY	Moyennes	Tableau des 30 moyennes d'élèves
T_Compte	Comptes	Tableau comptant le nombre de chaque lettre de 'A' à 'Z' du texte

Activité 2

Soit la séquence suivante :

```
A [ 1 ] ← 10
A [ 2 ] ← 7
A [ 3 ] ← A [ 1 ] Div A [ 2 ]
A [ 4 ] ← A [ 3 ]  A [ 2 ]
A [ 5 ] ← A [ 1 ] + A [ 3 ] * A [ 4 ]
```

1- Déclarer le tableau A.
2- Quel est le contenu de chaque élément du tableau A ?

Réponses

1- Déclaration

Tableau de déclaration de nouveaux types

Types
vecteur = tableau de 5 entiers

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
A	vecteur	Tableau de 5 éléments

2- Le contenu de chaque élément :

A	10	7	1	7	17
	1	2	3	4	5

Retenons

- Les structures de données vues dans cette leçon sont les tableaux (suite de variables de même type repérées par des indices).
- Avant d'utiliser un tableau, il faut le déclarer.
- Il faut bien faire la différence entre l'indice d'un élément et le type des éléments.

Exercices

Exercice 1

Elaborer un tableau de déclaration des objets relatifs aux :

- noms des employés
- leurs nombres de jours travaillés
- leurs salaires

d'une entreprise comptant 50 employés.

Exercice 2

1- Déclarer le tableau **JOURS** qui contient les sept jours de la semaine.

2- De même, déclarer le tableau **MOIS** permettant de regrouper les douze mois de l'année.

Exercice 3

Soit le tableau suivant :

T	14	10	19	84	92

1- Déclarer le tableau T en algorithmique et en Pascal.

2- Donner les affectations permettant de remplir T.

3- Inverser les éléments du tableau T sans utiliser un autre tableau.

Exercice 4

Soit le tableau de SALAIRES exprimés en Dinars.

SALAIRES	325.560	650.800	720.252	529.100	390.440
	1	2	3	4	5

1- Donner les cinq affectations permettant de remplir le tableau SALAIRES par les données précédentes.

2- Ranger, dans un deuxième tableau SALAIRES_MI, les équivalences des éléments de SALAIRES en millimes.

Exercice 5

1- Soit V et W deux tableaux d'entiers, de types respectifs T1 et T2 et de tailles respectives 4 et 3.

2- Soit la séquence d'affectations suivantes :

V[1] ← 300

V[2] ← 50

W[1] ← V[1] + V[2]* 2

V[3] ← W[1] DIV 3
W[2] ← W[1] MOD V[1]
W[3] ← V[2] * 2 + 2
V[4] ← (V[2] DIV 4) MOD 2
V[8] ← V[4]

Questions

1- Déclarer les deux tableaux V et W.
2- Trouver les erreurs d'affectation dans la séquence précédente.
3- Quel est le contenu de chaque élément des deux tableaux V et W ?

Exercice 6

Trouver les erreurs du programme Pascal suivant :

```
PROGRAM pleinerreurs ;
  CONST
       N=5 ;
  VAR
     I, J, N, M : INTEGER ;
     C: CHAR;
     A: ARRAY[1..N] of CHAR;
     B: ARRAY[1..M] of CHAR;
  BEGIN
     I:=0;
     C:='0';
     M:=4;
     A[1]:=C;
     A[2]:= SUCC(C);
     B[1]:=A[1];
  END.
```

Exercice 7

Soit le tableau **T** de mots dans lequel les mots sont classés par longueur (d'abord tous les mots d'une lettre, puis de deux lettres,…, puis de 8 lettres (taille maximale du mot)).
Un deuxième tableau, **CLE**, contient pour un indice i l'indice dans T du premier mot de longueur i. Que contient CLE[j] s'il n'y a pas dans T de mot de longueur j ? Donner un exemple contenant les déclarations et les affectations nécessaires.

CHAPITRE

Chapitre 2

Les actions élémentaires simples

Objectifs :

- Utiliser les structures simples pour résoudre des problèmes.
- Écrire des programmes en Pascal utilisant les structures simples.

Plan du chapitre :

Leçon 1

L'affectation

Objectifs spécifiques :

- Utiliser les structures simples pour résoudre des problèmes.
- Présenter les solutions sous forme d'un algorithme puis d'un programme.

Plan de la leçon :

I. Introduction

II. Définition

III. Vocabulaire et syntaxe

Retenons

Exercices

Leçon 1

L'affectation

J'entends par attribut ce que l'entendement perçoit d'une substance comme constituant son essence.

SPINOZA

I. Introduction

Dans cette leçon, nous allons revenir plus en détail sur l'action élémentaire intitulée **"affectation"**. Comme nous l'avons vu au chapitre précédent, l'affectation est l'action ou l'instruction élémentaire de base permettant de modifier la valeur d'une variable. Quand nous savons que la notion de variable est fondamentale dans l'exercice de la programmation, nous comprenons l'importance que peut prendre l'action d'affectation.

Activité 1

Soit la séquence d'instructions suivante :

1) X← 2
2) Y← X*X
3) Y← Y*Y
4) Y← Y*X
5) Y← Y*Y

1- Dresser un tableau (traces de la séquence algorithmique) pour déterminer les valeurs des variables X et Y après l'exécution des instructions précédentes.
2- Quel est le rôle de cette séquence d'instructions.

Réponse :

1- Trace de l'exécution des instructions précédentes

N° instruction	X	Y
1	2	
2	2	4
3	2	16
4	2	32
5	2	1024

2- Cette séquence d'instructions permet de calculer à partir d'une valeur X, la valeur Y correspondante. Après la dernière instruction, Y vaut X^{10}.

II. Définition

L'opération d'affectation consiste à attribuer une valeur à une variable. L'instruction d'affectation se note avec le symbole "←" en algorithmique et ":=" en Pascal.
L'expression à droite du symbole d'affectation sera évaluée puis le résultat sera affectée à la variable située à gauche du symbole "←".

Activité 2

Sachant que m, n, p et r sont quatre variables de type entier ou de type entier long. Soit la séquence ci-dessous :

1) m ← 159383552
2) n ← m DIV 8
3) p ← n DIV 1024
4) r ← p DIV (1024)

1- Déterminer la valeur de chacune des variables n, p et r après l'exécution de la séquence ci-dessus.
2- Que fait cette séquence d'instructions?
3- Traduire cet algorithme en Pascal.

Réponse :

1- - La valeur de n vaut 19922944.
- La valeur de p vaut 19456.
- La valeur de r vaut 19.

2- Cette séquence convertit une capacité mémoire donnée en bits en son équivalent en octets, kilo octets et méga octets.

Traduction en Pascal

```
PROGRAM CONVERSION;
USES WINCRT;
VAR
    m,n,p,r : LONGINT;
BEGIN
  m := 159383552;
  n := m DIV 8;
  p := n DIV 1024;
  r := p DIV 1024;
END.
```

Vous allez remarquer qu'en exécutant ce programme, vous n'allez rien voir à l'écran. Dans la leçon suivante, nous allons étudier comment afficher des méssages et des résultats à l'écran.

III. Vocabulaire et syntaxe

	Au niveau de l'analyse et de l'algorithme	Au niveau du Pascal
Forme générale	Variable ← expression	Variable := expression;

N.B. Une expression peut être une valeur.

Exemples :

Analyse et algorithme	Pascal	Commentaire
Y← 7	Y:=7;	La variable Y reçoit la valeur 7.
Ch ← "cible"	Ch : = 'cible';	La chaîne Ch reçoit la chaîne cible.
X ← Y	X : = Y;	La valeur de la variable X devient égale à la valeur de la variable Y.
L ← long(Ch) DIV 2	L←**length**(Ch) **DIV** 2;	La valeur de la variable L devient égale à la longueur de la chaîne Ch divisée par deux.
Y ← Y+1	Y : = Y+1;	La valeur de la variable Y devient égale à sa valeur actuelle incrémentée de 1.

Remarques :

- L'expression est évaluée avant d'être affectée à la variable.
- La valeur de l'évaluation de l'expression doit être compatible avec le type de la variable ; sinon l'exécution de cette affectation provoquera une erreur.
- Il est possible d'affecter la valeur d'une variable à une autre variable.
- Le nom d'une variable dans une expression signifie sa valeur actuelle.
- L'instruction d'affectation ne modifie que ce qui est situé à gauche du symbole d'affectation "←".

Activité 3

Effectuer une analyse, un algorithme et la traduction en Pascal du programme intitulé CRYPT, qui effectue le cryptage d'un mot donnée en utilisant le principe suivant :
- Permuter le premier caractère du mot avec le dernier.
- Modifier l'élément milieu du mot par son ordre dans le code ASCII.

NB. : On suppose que le mot est une chaîne de caractères dont la taille est supérieure à trois.

Analyse

	Nom = CRYPT	
S	**L.D.E.**	**O.U.**
11	**Résultat** = Ecrire (ch)	ch
10	Insère (c,ch,p)	p
9	Efface(ch,p,1)	c
8	Convch(ORD(Ch[p]),c)	l
7	p←(1+l) DIV 2	aux
6	Insère (aux, ch, 1)	
5	Efface(ch,1,1)	
4	ch[l] ← ch[1]	
3	aux ← sous-chaine(ch,l,1)	
2	l ← long(ch)	
1	ch = Donnée ("Entrer une chaîne de caractères: ")	
12	**Fin CRYPT**	

Tableaux de déclaration des objets

Objet	Type / Nature	Rôle
ch	Chaîne de Caractères	La chaîne saisie
p	Entier	La position du milieu de la chaîne
c	Chaîne de Caractères	Sauvegarde l'ordre du l'élément milieu de la chaîne dans le code ASCII
l	Entier	La longueur de la chaîne
aux	Chaîne de Caractères	Sauvegarde le premier caractère de la chaîne

Algorithme

0) Début CRYPT
1) Ecrire ("Entrer une chaîne de caractères: "), Lire(ch)
2) l ← long(ch)
3) aux ← sous-chaine(ch,l,1)
4) ch [l] ← ch[1]
5) Efface(ch,1,1)
6) Insère (aux,ch, ",1)
7) p←(1+l) DIV 2
8) Convch(ORD(Ch[p]),c)
9) Efface(ch,p,1)
10) Insère (c,ch,p)
11) Ecrire (ch)
12) Fin CRYPT

Traduction en Pascal

```
PROGRAM CRYPT;
USES WINCRT;
VAR  ch,aux,c: STRING;
     l,p:INTEGER;
BEGIN
  WRITE ('Entrer une chaîne de caractères: ');
  READLN(ch);
  l:=length(ch);
  aux:=copy(ch,l,1);
  ch[l]:= ch[1];
  Delete(ch,1,1);
  Insert(aux,ch,1);
  P:= (1+l) DIV 2;
  Str( Ord(Ch[p]),c);
  Delete(ch,p,1);
  Insert(c,ch,p);
  WRITE (ch);
END.
```

Un cas d'éxecution

```
Entrer une chaîne de caractères: info
o110fi
```

Retenons

- L'action d'affectation consiste à attribuer une valeur à une variable.
- L'instruction d'affectation se note avec le symbole "←".
- Une variable à affecter située à gauche de l'affectation doit avoir le même type ou un type compatible avec celui du résultat de l'évaluation de l'expression placée à droite du symbole d'affectation.

Exercices

Exercice 1

Soit la séquence d'instructions suivante :

X ← -5
X ← X*X
Y ← -X-3
Z ← (-X-Y)*3
X ← -(X+Y)*2+Z
Y ← Z*X*Y
Y ← -(Z+Y)
X ← X+Y-Z
Y ← X+Z
X ← Y+Z)/(X/10)
Y ← ((X*Z)/Y)*9

Dresser la trace de la séquence algorithmique précédente pour déterminer les valeurs des variables X,Y et Z après l'exécution des séquence d'instructions ci-dessus.

Exercice 2

1) Faire une analyse, écrire un algorithme puis la traduction en Pascal du programme intitulé PERMUT qui permet de permuter les contenus de deux variables X et Y en utilisant une variable auxiliaire.

2) Donner une deuxième méthode permettant de résoudre cet exercice sans l'utilisation d'une variable auxiliaire.

Exercice 3

Soit la séquence d'instructions suivante :

1) X ← 19.7
2) Y ← 114
3) K ← (X + Y - ABS(X-Y))/2
4) L ← (X + Y + ABS(X-Y))/2

1) Dresser la trace de la séquence algorithmique précédente pour déterminer les valeurs des variables K et L après l'exécution des instructions précédentes.

2) Quel est le rôle des instructions 3 et 4 ?

Exercice 4

Écrire un algorithme qui saisit un temps en seconde puis le convertit en jours, heure, minutes et secondes.

Exercice 5

On vous demande d'écrire un programme en Pascal qui permet de convertir une mesure d'énergie en Joule (J) saisie au clavier en son équivalent Decijoule (dJ), Hectojoule (hJ), Kilojoule (kJ), Calorie (cal), Kilocalorie (kcal), Wattheure (W/h), Kilowattheure (kWh), BTU.

Sachant que :

– 1 calorie = 4,1855 joules
– 1 kWh = 3 600 000 J
– 1 calorie ≈ 3,968 321 x 10^{-3} (BTU) British Thermal Unit

Exercice 6

Soit A un tableaux de 3 chaînes de caractères et B un tableaux de 2 entiers.

Soit la séquence d'affectation suivante :

- A[1] ← sous-chaine("communication",1,3)
- A[2] ← concat("sports",".", A[1])
- B[1] ← pos ("o", A[2])
- convch(2007, ch)
- A[3] ← "www."+ A[2]
- insérer(ch,A[3], 11)
- B[2] ← long(A[1])

Questions:

1) Quel est le contenu de chaque élément des deux tableaux A et B ?

2) Traduire cet algorithme en Pascal.

Exercice 7

On se propose d'écrire un programme en Pascal intitulé POIDS qui calcule le poids d'une image numérique prise par un appareil photo numérique de sept mégapixel de résolution et d'un codage de 6 octets (48 bits/pixe).

NB. Le poids d'une image est le produit du nombre de pixels de l'image par le nombre d'octets par pixel.

Exercice 8

Écrire un programme en Pascal qui permet de calculer la vitesse de rotation de la terre autour du soleil qui est exprimée en km/s.

NB. - La distance moyenne Terre-Soleil est de l'ordre de 150 000 000 km.
- La vitesse de rotation de la terre autour du soleil = corconférence de l'orbite/ (365 jours*24heures*3600 secondes).
- La corconférence de l'orbite= 2*π * distance moyenne Terre-Soleil

Leçon 2

Les operations d'entree / sortie

Objectifs spécifiques

- Savoir faire une lecture et une écriture de données.
- Présenter les solutions sous forme d'un algorithme puis d'un programme.

Plan de la leçon

Leçon 2

Les operations d'entree / sortie

Conduire par ordre mes pensées, en commencant par les objets les plus simples et les plus aisés à connaitre, pour monter peu à peu, comme par degrés, jusqu'à la connaissance des plus composés.

DESCARTES

Nous avons vu dans la leçon précédente qu'une opération d'affectation consiste à attribuer une valeur à une variable. Cette opération se fait au niveau de la mémoire et il est intéressant de visionner le résultat de ces opérations. Nous verrons aussi comment permettre à l'utilisateur de saisir des données à mettre dans des variables par l'intermédiaire du clavier. Ces deux actions sont convenues élémentaires et s'intitulent respectivement sortie de données et entrée de données.

I. L'opération de sortie

Activité 1

Effectuer une analyse, écrire un algorithme et faire la traduction en Pascal du programme intitulé DISTANCE qui convertit une distance n = 24 pouces en son équivalent pieds et mètres. Afficher les résultats.

On rappel que :

- 1 pied =12 pouces
- 1 pouce = 2.54 cm

Analyse

	Nom = DISTANCE	
S	**L.D.E.**	**O.U.**
4	**Résultat** = Ecrire("La mesure en mètre vaut : ",m)	m
	Ecrire("La mesure en pied vaut : ",p)	p
3	m ← (n* 2.54) / 100	n
2	p ← n div 12	
1	n ← 24	
5	**Fin DISTANCE**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
m	Réel	Mesure en mètre
p	Réel	Mesure en pied
n	Entier	Mesure en pouce

Algorithme

0) Début DISTANCE
1) n ← 24
2) p ← div 12
3) m ← (n 2.54) / 100*
4) Ecrire("La mesure en mètre vaut : ",m)
Ecrire("La mesure en pied vaut : ",p)
5) Fin DISTANCE

Traduction en Pascal

```
PROGRAM DISTANCE;
USES WINCRT;
VAR
  m,p:REAL;
  n:INTEGER;
BEGIN
  n :=24 ;
  p :=n DIV 12;
  m := (n* 2.54) / 100;
  WRITELN('La mesure en pied vaut :',p);
  WRITELN('La mesure en mètre vaut :',m);
 END.
```

I.1 Définition

La sortie de données est l'action convenue élémentaire qui consiste à écrire une donnée sur un périphérique de sortie tel que l'écran, l'imprimante, etc.

Activité 2

Effectuer une analyse, écrire un algorithme et faire la traduction en Pascal du programme intitulé INVERSE qui saisit un entier de trois chiffres puis détermine le nombre correspondant lorsqu'on l'écrit à l'envers.

Exemple : *123 devient 321*
et 120 devient 21

Analyse

	Nom = INVERSE	
S	**L.D.E.**	**O.U.**
6	**Résultat** = Ecrire (I)	I
5	I ← C+(D*10)+(U*100)	C
4	U ← E MOD 10	D
3	D ← (E DIV 10) MOD 10	U
2	C ← E DIV 100	E
1	E = Donnée("Donner un entier naturel de trois chiffres ")	
7	**Fin INVERSE**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
I	Entier	L'inverse de E
C	Entier	Chiffre des centaines de E
D	Entier	Chiffre des dizaines de E
U	Entier	Chiffre des unités de E
E	Entier	L'entier E

Algorithme

0) Début INVERSE
1) Lire (E)
2) C ← E DIV 100
3) D ← (E DIV 100) MOD 10
4) U ← E MOD 10
*5) I ← C+(D*10)+(U*100)*
6) Ecrire (I)
7) Fin INVERSE

Traduction en Pascal

```
PROGRAM INVERSE;
USES WINCRT;
VAR
   E,U,D,C,M,I : INTEGER;
BEGIN
   WRITE('Donner un entier naturel de trois chiffres : '); READLN(E);
   C:= (E DIV 100);
   D:= (E DIV 10) MOD 10;
   U:= E MOD 10;
   I:= C+(D*10)+(U*100);
   WRITE (I);
END.
```

Un cas d'éxecution

```
Donner un entier naturel de trois chiffres : 236
632
```

I.2 Vocabulaire et syntaxe

Au niveau de l'analyse et de l'algorithme	Au niveau de Pascal	Commentaire
Ecrire(nom_variable)	**WRITE** (nom_variable); **WRITELN** (nom_variable);	Affichage du contenu de la variable identifiée par nom_variable
Ecrire("message")	**WRITELN** ('message') ;	Affichage de la chaîne de caractères "message" telle que vous l'avez écrite.
Ecrire("message",nom_variable)	**WRITELN**('message', nom_variable) ;	La valeur de la variable X devient égale à la valeur de la variable Y.
Ecrire (Expression)	**WRITELN** (expression);	La valeur de la variable L devient égale à la longueur de la chaîne Ch divisée par deux.

Exemple 1 :

Analyse et algorithme
Ecrire(long*larg)
Ecrire("la surface d'un rectangle est : ", long*larg)
Ecrire("la surface d'un rectangle est : ", long ,"* ",larg,"= ", s)

Exemple 2 :

Analyse et algorithme	Pascal
Ecrire (age)	**WRITE** (age);
Ecrire (Nom," ",prénom)	**WRITELN** (Nom,' ',prénom);
Ecrire (Nom," ",prénom," a pour age : ",age)	**WRITELN** (Nom,' ',prénom,' a pour age : ' ,age);

Remarques :

- En Pascal l'affichage d'un texte est placé entre apostrophes.
- La procédure WRITE affiche et laisse le curseur juste après le dernier caractère affiché.
- La procédure WRITELN provoque un retour à la ligne après l'affichage.

I.3 Formatage de l'affichage des résultats

I.3.1. Introduction

Revenons à l'activité 1 et observons les résultats obtenus.

Affichage des données	Affichage des résultats
- Mesure en pouce est 1000	- Mesure en pied vaut 8.3000000000E+01 - Mesure en mètre vaut 2.5400000000E+01

```
Entrer une mesure en pouce: 1000
La mesure en pied vaut :  8.3000000000E+01
La mesure en mètre vaut :  2.5400000000E+01
```

Dans le langage Pascal, l'écran est par défaut, divisé en colonnes de largeur de 10 caractères, l'affichage des nombres se fait à partir de la droite de la colonne en cours. Cependant, l'utilisateur pourra imposer son format d'écriture en utilisant des facilités offertes par le langage. On pourra fixer la largeur de la colonne ainsi que le nombre de décimales s'il s'agit de l'écriture d'un réel.

Pour chacun des activités suivantes, interpréter les différents affichages et écrire le programme en Pascal en effectuant vous-même les modifications.

I.3.2. Affichage des entiers

Activité 3

```
PROGRAM Format1;
USES WinCrt;
CONST Entier1 = 114;
      Entier2 = -9999999;
BEGIN
  WRITELN ('...Test sur les entiers...');
  WRITELN ('12345678901234567890123456789O');
  WRITELN ('------------------------------');
  WRITELN ( Entier1:2);
  WRITELN ( Entier1:3);
  WRITELN ( Entier1:5);
  WRITELN ( Entier1:24, Entier2);
END.
```

L'exécution donne :

```
...Test sur les entiers...
12345678901234567890123456789O
------------------------------
114
114
  114
                        114-9999999
```

Interprétation :

- L'instruction **WRITELN** (Entier1:5) permet d'afficher la valeur de l'entier Entier1 sur 5 caractères.

- Lorsque le nombre de caractères que l'on précise pour l'affichage est supérieur ou égal à ce qui est requis, la valeur est affichée en étant justifiée à droite.

- Si on fournit un nombre de caractères inférieur à ce qui est requis pour l'affichage alors le compilateur n'en tient pas compte et affiche le résultat sur un nombre correct de caractères.

Syntaxe :

- **WRITE** (valeur_entière : n) affiche la valeur entière dans une colonne de n caractères à partir de la droite. Si la valeur entière comporte plus que n chiffres alors l'affichage commence par la gauche.

I.3.3. Affichage des réels

Activité 4

```
PROGRAM Format2;
USES WinCrt;
CONST Reel1   = 19.0;
      Reel2   = 1.23456E-3;

BEGIN
  WRITELN ('...Test sur les réels...');
  WRITELN ('12345678901234567890123456789O');
  WRITELN ('------------------------------');
  WRITELN ( Reel1);
  WRITELN ( Reel1:5:0);
  WRITELN ( Reel1:2:3);
  WRITELN ( Reel1:7:1);
  WRITELN ('------------------------------');
  WRITELN ( Reel2);
  WRITELN ( Reel2:10:5);
  WRITELN ( Reel2:2:5);
END.
```

L'exécution donne :

```
...Test sur les réels...
12345678901234567890123456789O
------------------------------
 1.9000000000E+01
   19
19.000
   19.0
------------------------------
 1.2345600000E-03
   0.00123
0.00123
```

Interprétation :

- Pour les nombres réels, la syntaxe est étendue puisque nous voyons apparaître un second ":" suivi d'un nombre. La première séquence a la même signification qu'avec les entiers ; par exemple, **WRITELN** (Réel1 :5 :0) ; provoquera l'affichage de ce nombre sur 5 caractères (si c'est possible).
- La seconde séquence précise le nombre de décimales que nous désirons afficher pour le nombre en question.
- Si vous ne précisez pas le nombre de décimales alors le compilateur optera pour la notation scientifique en puissance de 10.

Syntaxe :

- **WRITE** (valeur_réelle) affiche le nombre en notation scientifique (x.xxxxxE+xx précédé d'un espacement.)
- **WRITE** (valeur_réelle : np) affiche le nombre en notation scientifique sur np positions précédé d'un espacement.
- **WRITE** (valeur_réelle : np : nd) affiche le nombre sur np positions avec nd décimales.

I.3.4. Affichage des chaînes de caractères

Activité 5

```
PROGRAM Format3;
USES WinCrt;
CONST   Ch1 = 'Pêche';
        Ch2 = 'sous-marine' ;
```

```
BEGIN
  WRITELN ('...Test sur les chaines de caracteres...');
  WRITELN ('12345678901234567890123456789O');
  WRITELN ('------------------------------');
  WRITELN ( Ch1);
  WRITELN ( Ch1:5);
  WRITELN ( Ch1:7);
  WRITELN ( Ch1,Ch2);
  WRITELN ( Ch1:7,Ch2:4);
  WRITELN ( Ch1:7,Ch2:12);
 END.
```

L'exécution donne :

```
...Test sur les chaines de caracteres...
12345678901234567890123456789O
------------------------------
Pêche
Pêche
  Pêche
Pêchesous-marine
  Pêchesous-marine
  Pêche sous-marine
```

Syntaxe :

L'affichage d'une chaîne de caractères se fait normalement à la place du curseur.

- **WRITE** (chaîne : n) affiche la chaîne sur n positions : insertion d'espacement à gauche de la chaîne si il y a moins de n caractères sinon si n est insuffisant alors ajustement automatique.

I.3.5. Affichage des caractères

Activité 6

```
PROGRAM Format4;
USES WinCrt;
CONST   Char1   = 'X';
        Char2   = 'Y';

BEGIN
  WRITELN ('...Test sur les caracteres...');
  WRITELN ('12345678901234567890123456789O'
```

```
  WRITELN ('-----------------------------');
  WRITELN ( Char1);
  WRITELN ( Char1:2);
  WRITELN ( Char1:3, Char2:4);
  WRITELN ( Char1:3, Char2:5);
END.
```

L'exécution donne :

```
...Test sur les caracteres...
12345678901234567890123456789
-----------------------------
X
 X
  X   Y
  X    Y
```

Syntaxe :

L'affichage d'un caractère se fait normalement à la place du curseur.

- **WRITE** (car : n) affiche le caractère à la position n et insertion d'espacement à gauche du caractère.

Remarque :

Formater les sorties signifie qu'on désire leur imposer un format d'affichage.

Activité 7

Écrire en Pascal le programme intitulé ConversionDinarEuro, qui convertit un montant en dinars en son équivalent en Euro et inversement.

Traduction en Pascal

```
PROGRAM ConversionDinarEuro;
USES WinCrt;
CONST
   Affichage = 8; (* nombres affiches sur 8 caractères *)
   Precision = 3; (* 2 décimales *)
   Change    = 1.705;
VAR
   PrixDinar, PrixEuro : real;
BEGIN
  WRITE ('Entrez un prix en Dinars : ');
  READLN (PrixDinar);
  WRITE ('Entrez un prix en Euros : ');
```

```
 READLN (PrixEuro);
 WRITE (Prixdinar:Affichage:Precision);
WRITELN ('D = ', (PrixDinar/Change):Affichage:Precision, 'E');
 WRITE (PrixEuro:Affichage:Precision);
 WRITELN ('E = ', (PrixEuro*Change):Affichage:Precision, 'D');
END.
```

II. Les opérations d'entrée

Activité 8

Un son numérique est caractérisé par :

- la qualité. Elle est déterminée par la fréquence d'échantillonnage c-à-d plus la fréquence est élevée, plus on prélève d'échantillons et par conséquent, meilleure sera la qualité.
- la résolution. Elle est déterminée par la plage de valeurs que peuvent prendre les échantillons, cette plage est caractérisée par le nombre de bits alloués. Plus on alloue de bits, plus la plage est grande et plus l'information sur le son est abondante.
- Le nombre de voies (monophonique, stéréophonique …) .

On vous demande de déterminer et d'afficher la taille occupée par un morceau de musique ayant les caractéristiques suivantes :

- Durée : 140 secondes
- Taux d'échantillonnage : 22000 hertz
- Résolution : 16 bits
- Son stéréophonique

1) Effectuer une analyse du problème intitulé MUSIQUE.
2) Déduire l'algorithme de cette analyse.
3) Traduire en Pascal cet algorithme.

Analyse

	Nom = MUSIQUE	
S	**L.D.E.**	**O.U.**
6	**Résultat** = Ecrire ("la taille de cette musique en MO est : ",T)	T
5	T ← F*R*V*D DIV (1024*1024*8)	V
4	V = Donnée ("Entrer le nombre de voies: ")	R
3	R = Donnée ("Entrer la résolution en bits: ")	F
2	F = Donnée ("Entrer la fréquence d'échantillonnage en HZ: ")	D
1	D = Donnée ("Entrer la durée en secondes : ")	
7	**Fin MUSIQUE**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
T	Entier	La taille de la musique
V	Entier	Le nombre de voies
R	Entier	La résolution
F	Entier	Fréquence d'échantillonnage
D	Entier	La durée d'enregistrement

Algorithme

0) Début MUSIQUE
1) Ecrire ("Entrer la durée en secondes : "), Lire(D)
2) Ecrire ("Entrer la fréquence d'échantillonnage en HZ : "), Lire(F)
3) Ecrire ("Entrer la résolution en bits : "), Lire(R)
4) Ecrire ("Entrer le nombre de voies : "), Lire(V)
*5) T← F*R*V*D* ***DIV*** *(1024*1024*8)*
6) Ecrire ("la taille de cet musique en MO est : ",T)
7)Fin MUSIQUE

Traduction en Pascal

```
PROGRAM MUSIQUE;
USES WINCRT;
VAR
   D,V,R,F,T: LONGINT;{*entier long sur 4 octéts*}

BEGIN
 WRITELN('Entrer le temps d''enregistrement en secondes : ');
 READLN(D);
 WRITELN('Entrer la fréquence d''échantillonnage en HZ: ');
 READLN(F);
 WRITELN('Entrer la résolution en bits: '); READLN(R);
 WRITELN('Entrer le nombre de voies: '); READLN(V);
 T:= F*R*V*D DIV (1024*1024*8);
 WRITELN('la taille de cette musique en MO est : ',T);
END.
```

Un cas d'exécution

```
Entrer le temps d'enregistrement en secondes :
45
Entrer la fréquence d'échantillonnage en HZ:
22000
Entrer la résolution en bits:
16
Entrer le nombre de voies:
2
la taille de cette musique en MO est : 3
```

II. 1 Définition

Une entrée consiste à introduire une donnée à partir d'une source d'entrée (clavier, souris, stylo optique …). C'est une opération qui permet d'affecter à une variable en mémoire, une valeur de même type ou compatible avec celle de la variable.

Activité 9

On se propose d'écrire un programme intitulé IMAGE qui calcule le nombre de pixels et le poids d'une image numérique en méga octet (MO), d'une dimension (hauteur, largeur) exprimées en pouces et d'une résolution en dpi (points par pouce).
Analyser ce problème et en déduire l'algorithme et le programme Pascal correspondant.

NB.

- La résolution indique le nombre de points sur une unité de longueur de l'image.
- Le nombre de pixels dans une image = hauteur * résolution* longueur *résolution
- L'image est codée en 24 bits/pixel ou 48 bits/pixel c'est à dire 8 ou 16 bits par canal R (rouge), V (vert) et B (bleu).
- Le nombre total d'octets dans l'image (poids) = Nombre de pixels * nombre d'octets par pixel .

Exemple :

Soient les caractéristiques suivantes d'une image :
- largeur 4 pouces
- hauteur 5 pouces
- résolution 1200 dpi
- codage de 3 octets en 24 bits (1octet par canal R,V et B).
  - Le nombre de pixels dans cette image est : $4*5*1200^2$
  - Le poids de cette image est : $4*5*1200^2*3$ octets

Analyse **Nom = IMAGE**

S	L.D.E.	O.U.
8	**Résultat** = Ecrire(n,p1)	n
7	P1 ← p / (1024*1024)	p1
6	p ← n*c	c
5	n ← l*r*h*r	p
4	c=DONNEE ("Entrer le codage de l'image : ")	r
3	r=DONNEE("Entrer la résolution de l'image: ")	h
2	h=DONNEE ("Entrer la hauteur de l'image : ")	l
1	l=DONNEE("Entrer la largeur de l'image: ")	
9	**Fin IMAGE**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
n	Entier	Nombre de pixels
pl	Entier	Poids de l'image en méga octet
c	Entier	Codage de l'image
p	Entier	Poids de l'image en octet
r	Entier	Résolution de l'image
h	Entier	Hauteur de l'image
l	Entier	Largeur de l'image

Algorithme

0) Début IMAGE
1) Ecrire("Entrer la largeur de l'image: "); Lire(l)
2) Ecrire("Entrer la hauteur de l'image: "); Lire (h)
3) Ecrire("Entrer la résolution de l'image: "); Lire(r)
4) Ecrire("Entrer le codage de l'image: "); Lire(c)
*5) n ← l*r*h*r*
*6) p ← n*c*
*7) P1 ← p / (1024*1024)*
8) Ecrire(n,p1)
9) Fin IMAGE

Traduction en Pascal

```
PROGRAM IMAGE;
USES WINCRT;
VAR
   n,c,p,r,h,l:INTEGER;
   P1:REAL;
BEGIN
  WRITELN ('Entrer la largeur de l''image:');READLN(l);
  WRITELN ('Entrer la hauteur de l''image : '); READLN (h);
  WRITELN ('Entrer la résolution de l''image: '); READLN (r);
  WRITELN ('Entrer le codage de l''image: '); READLN (c);

  n:= l*r*h*r;
  p:= n*c;
  P1:= p / (1024*1024);
  WRITE(n,p1);

END.
```

II. 2 Vocabulaire et syntaxe

Niveau analyse	Niveau algorithme	Niveau de Pascal
Variable = donnée	Lire (variable)	**READLN** (variable); ou **READ** (variable);
var1,var2,var3 = donnée	Lire (var1,var2,var3)	**READLN** (var1,var2,var3); ou **READ** (var1,var2,var3);

Exemples :

Analyse	Algorithme	Pascal
X = donnée	Lire(X)	**READLN** (X);
Y= donnée ("Entrer Y")	Lire (variable)	**READLN** (variable); ou **READ** (variable);
(ch1,ch2,ch3) =donnée	Lire (var1,var2,var3)	**READLN** (var1,var2,var3); ou **READ** (var1,var2,var3);

Remarques :

- Le langage PASCAL comprend deux procédures standards de lecture : **READ** et **READLN**.

- Les instructions **READ** et **READLN** permettent de transférer des données vers des variables situées dans la mémoire centrale à travers des périphériques d'entrée. Par défaut, le périphérique d'entrée utilisé est le clavier.

- Lorsqu'on utilise une seule instruction **READLN** pour lire plusieurs variables, on fait entrer les valeurs séparées par des virgules. Cependant, dans le cas où chaque variable est lue séparément, un retour de chariot est nécessaire, après chaque valeur entrée à partir du clavier, pour déclencher la lecture de celle-ci.

- La lecture de plusieurs variables de type Chaîne à la fois est non fonctionnelle.

- La seule différence qui existe entre **READLN** et **READ**, c'est que **READLN** commande un changement de ligne après la lecture de la liste des variables spécifiées comme paramètres. Ce changement de ligne ne prend effet qu'à la prochaine lecture occasionnée par un autre énoncé **READ** ou **READLN**.

Activité 10

Écrire en Pascal le programme intitulé DATE, qui initialise la date maintenue par le système d'exploitation, en utilisant la procédure SetDate (…) de la bibliothèque WinDos.

NB.

- La syntaxe de la procédure est : procedure SetDate(Annee, Mois, jour);
- les paramètres valides sont 1980..2099 pour Annee, 1..12 pour Mois et 1..31 pour Jour. Si la date est invalide, la demande est ignorée.

Traduction en Pascal

```
PROGRAM DATE;
USES WinCrt,WinDos;
VAR
   a,m,j: WORD;

BEGIN
 WRITE ('Entrez l''année : ');READLN ( a);
 WRITE ('Entrez le mois : '); READLN ( m);
 WRITE ('Entrez le jour : '); READLN ( j);
 SetDate(a,m,j);
END.
```

Remarque :

le type WORD est le type entier non signé ; le domaine de définition est de 0 à 65535.

Retenons

– Pour afficher à l'écran une donnée quelconque, on utilise l'instruction élémentaire de sortie traduite au niveau de l'algorithme et de l'analyse par le verbe Erire et en Pascal par l'instruction WRTIE ou WRITELN.

– Pour imposer un format d'affichage sur l'écran on doit formater les sorties. Cependant, l'utilisateur pourra imposer son format d'écriture en fixant la largeur de la colonne ainsi que le nombre de décimales s'il s'agit de l'écriture d'un réel.

– Pour saisir une donnée via le clavier pour la mettre dans une variable donnée, on utilise l'instruction élémentaire d'entrée exprimée par le verbe Lire ou le mot DONNÉE et en Pascal par l'instruction READ ou READLN.

Exercices

Exercice 1

Écrire un algorithme permettant de calculer et d'afficher la durée entre deux horaires exprimés en heures, minutes, secondes. Cette durée sera exprimée aussi en heures, minutes et secondes.

Exercice 2

Faire une analyse, déduire un algorithme puis le traduire en un programme Pascal intitulé SOMME qui calcule la somme des n premiers entiers naturels non nuls en utilisant la formule mathématique suivante : 1+2+…..+n =(n*(n+1))/2.

Exercice 3

X et Y étant deux valeurs numériques. Écrire une analyse, un algorithme et la traduction en Pascal du programme intitulé CALCUL qui effectue la somme, le produit et la moyenne arithmétique de X et Y.

Exercice 4

La puissance d'une installation de pompage est de 100 kW.
Le rendement de cette installation est de 75%.
Effectuer une analyse, un algorithme et la traduction en Pascal du programme intitulé POMPAGE qui permet de calculer le débit exprimé en litres d'eau par seconde?
Sachant que :

- La puissance utile de l'installation = La puissance de l'installation * Le rendement de cette installation.
- Pour remonter un litre d'eau à la surface, il faut produire un travail W = Force*profondeur (m).
- La force = masse d'un litre d'eau (1Kg) * pesanteur G(10N /Kg).
- Le débit de la pompe en litres par seconde = La puissance utile de l'installation/le travail W.

Exercice 5

Deux trains, distants de 60 km, roulent l'un vers l'autre sur le même itinéraire.
Le train A avance à une vitesse de 70 km/h.
Le train B avance à une vitesse de 55 km/h.
Faire une analyse du programme intitulé RENCONTRE qui permet de calculer le temps où les deux trains vont se rencontrer?

Exercice 6

Pour créer un répertoire, on utilise la procédure prédéfinis MkDir(Path: string) de l'unité DOS.
Écrire un programme en Pascal, qui permet de créer un répertoire ; le nom du répertoire et le chemin sont saisis au clavier
Exemple : MkDir('C:\TEST'); permet de créer le répertoire 'test' sous la racine C :.

Exercice 7

Écrire un programme qui demande à l'utilisateur les coordonnées de deux points distincts du plan et qui affiche le coordonnées du point milieu.

Exercice 8

Préciser le résultat obtenu à la sortie d'un programme par les séquences d'instructions suivantes :

```
VAR  var1, var2, var3, var4, var5 : REAL;
```

où :
var1 vaut 1.2
var2 vaut 0.00089
var3 vaut 45
var4 vaut 10095.095
var5 vaut 2.7182818281

Séquences d'instructions :

```
1) WRITE(var1 : 8 : 4);        6) WRITE(var1 : 6 : 4);
2) WRITE(var2 : 8 : 4);        7) WRITE(var2 : 10 : 6);
3) WRITE(var3 : 4 : 1);        8) WRITE(var3 : 10 : 6);
4) WRITE(var4 : 3 : 1);        9) WRITE(var4 : 7 : 5);
5) WRITE(var5 : 3 : 1);        10) WRITE(var5 : 10 : 4);
```

Exercice 9

Écrire un programme en Pascal permettant de lire 2 nombres entiers positifs inférieurs à 999 et affiche à l'écran la multiplication et la division comme suit :

```
   120                          130 : 11
 x  11                           9  :.....
 ......                             : 11
 = 1320                             :
```

Exercice 10

Écrire un programme qui demande à l'utilisateur une valeur pour U_0, r et n et qui affiche la nième valeur de la suite arithmétique définie par U_0 et $U_{n+1} = U_n + r$.
(On rappelle la propriété : $U_n = U_0 + n.r$)

Exercice 11

Écrire un programme en Pascal qui saisit une capacité d'un disque dur puis en reprenant les caractéristiques ci-dessus, calculer et afficher :

- le nombre de disquettes nécessaires à une sauvegarde complète du disque dur, celui-ci étant supposé entièrement rempli.
- Si on utilise des CDROM au lieu des disquettes, donner le nombre de CD.
- Si on utilise un logiciel de compression de données avec un taux de compression 20%, quel sera le nombre de CD nécessaires à une sauvegarde complète du disque.

CHAPITRE

Chapitre 3

Les structures de contrôle conditionnelles

Objectifs :

- Résoudre des problèmes faisant appel aux structures de contrôle conditionnelles.
- Choisir la forme adéquate des structures de contrôle conditionnelles.
- Présenter les solutions sous forme d'un algorithme puis d'un programme.

Plan du chapitre :

Leçon 1

La structure de contrôle conditionnelle simple

Objectifs spécifiques :

- Résoudre des problèmes faisant appel aux structures de contrôle conditionnelles simples.
- Présenter les solutions sous forme d'un algorithme puis d'un programme.

Plan de la leçon :

Leçon 1

La structure de contrôle conditionnelle simple

L'art est une certaine disposition, accompagnée de règle vraie, capable de produire; le défaut d'art, au contraire, est une disposition à produire accompagnée de règle fausse.
ARISTOTE

I. Introduction

Vous avez vu en 3ème année que les structures de contrôle conditionnelles permettent à un ordinateur de prendre des décisions. En effet, les opérations à accomplir peuvent varier dans un algorithme selon les données fournies.

Par exemple :
- Aviser la date de validité d'un logiciel en version démonstration.
- Calculer le montant net d'une facture en fonction des remises accordées.
- Déclencher une alarme en fonction de l'état d'un détecteur.
- Donner les propriétés d'un nombre donné (pair, impair, premier …).
- Evaluer des sorties en utilisant des fonctions logiques.
- Déterminer la nature d'un caractère tapé au clavier.

Il existe trois types de structures de contrôle conditionnelles :
- la structure conditionnelle simple
- la structure conditionnelle généralisée
- la structure conditionnelle à choix

Dans cette leçon, nous expliquerons comment écrire des algorithmes répondant à la structure de contrôle conditionnelle simple.

II. La structure de contrôle conditionnelle simple

II.1 La forme réduite

II.1.1. Vocabulaire et syntaxe

Au niveau de l'analyse

Décision = [Initialisation] Si condition Alors
Instruction 1
Instruction 2
- - - - -
- - - - -
Instruction n
FinSi

} Ensemble d'instructions exécutées si la condition est valide

Remarques :

- L'initialisation et le traitement sont généralement des séquences formées de plusieurs instructions. Nous obtenons donc l'écriture suivante :

```
                Init i1
                - - -
Décision =      - - -      Si condition Alors
                Init iP
                                Instruction 1
                                Instruction 2
                                - - - - -        } Traitement
                                - - - - -
                                Instruction n
                           FinSi
```

- La **condition** est une expression logique qui prend soit la valeur « VRAI » soit la valeur « FAUX».
- Si la condition est « VRAI », les instructions entre Alors et FinSi sont exécutées.
- La condition peut être une condition composée de plusieurs autres propositions logiques liées par les opérateurs booléens.

  Exemple : Not (n>2) OU (x>3) ET (x<50)

Au niveau de l'algorithme

```
- - -
{Initialisation} Si condition   Alors
                                  Instruction 1
                                  Instruction 2
                                  - - - - -      } Traitement
                                  - - - - -
                                  Instruction n
                               FinSi
```

Traduction en Pascal

```
-   -   - ;
Inst1; InstP;    {le corps d'initialisation}
IF condition THEN
                 BEGIN
                   Instruction i1;
                   Instruction i2;
                   -   -   -   - ;     } Traitement
                   -   -   -   - ;
                   Instruction in;
                 END;
-   -   - ;
```

Activité 1

On se propose d'écrire un programme intitulé REGULATEUR qui affiche le message "Régulateur déclenché" et produit un bip sonore si la température n'est pas comprise entre 18 °C et 24 °C.
Analyser ce problème et en déduire l'algorithme correspondant.

NB. On rappelle que CHR(7) produit un bip sonore.

Analyse

	Nom = REGULATEUR	
S	**L.D.E.**	**O.U.**
3	**Résultat** = Ecrire (Bip,Message)	Bip Message Temp
2	(Bip,Message) = [Bip← "", Message ← "Régulateur non déclenché"] **Si** (Temp < 18) ou (Temp > 24) **Alors** Message ← " Régulateur déclenché" Bip ← CHR(7) **FinSi**	
1	Temp = Donnée ("Entrer la valeur de la température : ")	
4	**Fin REGULATEUR**	

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
Bip	Caractère	Produit un bip sonore
Message	Chaîne de caractères	Sauvegarde l'état de l'alarme
Temp	Entier	Saisie la température

Algorithme

0) Début REGULATEUR
1) Écrire ("Entrer la valeur de la température : "), Lire (Temp)
2) [Bip ← "", Message ← "Régulateur non déclenché"]
Si (Temp < 18) ou (Temp > 24) Alors
Message ← "Régulateur déclenché"
Bip ← chr(7)
FinSi
3) Ecrire (Bip,Message)
4) Fin REGULATEUR

II.1.2. Définition

Une structure de contrôle conditionnelle est dite à forme simple réduite lorsque le traitement dépend d'une condition. Si la condition est évaluée à « VRAI », le traitement est exécuté.

Activité 2

Traduire l'algorithme REGULATEUR en Pascal.

Traduction en Pascal

```
PROGRAM REGULATEUR;
USES WINCRT;
VAR
  Temp : INTEGER;
  Message : STRING;
  Bip : CHAR;

BEGIN
WRITE ('Entrer la valeur de la température : ');
READLN(Temp);
Bip := '';
Message := 'Régulateur non déclenché';
IF(Temp < 18) OR (Temp > 24)
           THEN
             BEGIN
               Message := 'Régulateur déclenché';
               Bip := Chr(7);
             END;
WRITE(Bip,Message);
END.
```

Activité 3

On se propose d'écrire un programme intitulé CLASSEMENT qui saisit 2 chaînes de caractères puis les affiche en commençant par la plus longue. Si elles sont de même longueur la première saisie sera affichée en premier.

Questions :

1) Analyser ce problème et en déduire l'algorithme correspondant.

2) Traduire l'algorithme CLASSEMENT en Pascal.

Tableaux de déclaration des objets

	Nom = CLASSEMENT	
S	**L.D.E.**	**O.U.**
4	**Résultat** = Ecrire (Chaine1, Chaine2)	Chaine1 Chaine2
3	(Chaine1, Chaine2) = [Chaine1 ← Ch1, Chaine2 ← Ch2] **Si** (LONG(Ch2) > LONG(Ch1)) **Alors** Chaine1 ← Ch2 Chaine2 ← Ch1 **FinSi**	Ch1 Ch2
2	Ch2 = Donnée ("Entrer la Chaine 2 : ")	
1	Ch1 = Donnée ("Entrer la Chaine 1 : ")	
5	**Fin CLASSEMENT**	

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
Chaine1	Chaîne de Caractères	Sauvegarde la plus petite chaîne
Chaine2	Chaîne de Caractères	Sauvegarde la plus longue chaîne
Ch1	Chaîne de Caractères	Saisie la première chaîne
Ch2	Chaîne de Caractères	Saisie la deuxième chaîne

Algorithme

1) Début CLASSEMENT
2) Ecrire ("Entrer la Chaîne 1 : "), Lire (Ch1)
3) Ecrire ("Entrer la Chaîne 2 : "), Lire (Ch2)
4) [Chaine1 ← Ch1, Chaine2 ← Ch2] Si (LONG(Ch2) > LONG(Ch1)) Alors
Chaine1 ← Ch2
Chaine2 ← Ch1
FinSi
4) Ecrire (Chaine1, " ", Chaine2)
5) Fin CLASSEMENT

Traduction en Pascal

```
PROGRAM CLASSEMENT;
USES WINCRT;
VAR
     Chaine1,Chaine2,Ch1,Ch2:STRING;
```

```
BEGIN
     WRITE ('Entrer la Chaîne 1 : '); READLN(Ch1);
     WRITE ('Entrer la Chaîne 2 : '); READLN(Ch2);
     Chaine1 := Ch1;
     Chaine2 := Ch2;

     IF (LONG(Ch2) > LONG(Ch1)) THEN
                                BEGIN
                                  Chaine1 := Ch2;
                                  Chaine2 := Ch1;
                                END;
    WRITE (Chaine1,' ', Chaine2);
END.
```

II.2 La forme alternative

II.2.1. Vocabulaire et syntaxe

Au niveau de l'analyse et de l'algorithme

Décision = [Initialisation] **Si** condition **Alors**

Instruction 1.1
Instruction 1.2
- - - - -
- - - - -
Instruction 1.n

} Traitement 1 : Ensemble d'instructions exécutées si la condition est valide

Sinon

Instruction 2.1
Instruction 2.2
- - - - -
- - - - -
Instruction 2.m

} Traitement 2 : Ensemble d'instructions exécutées si la condition est fausse

FinSi

Remarques :

- Lorsque l'évaluation de la condition produit la valeur :
  VRAI seules les instructions du traitement 1 sont exécutées.
  FAUX seules les instructions du traitement 2 sont exécutées.
- Chaque traitement peut comporter une ou plusieurs instructions.
- Les instructions du traitement 1 sont délimitées entre « Alors » et « Sinon » et celles du traitement 2 le sont entre « Sinon » et « Finsi ».

Traduction en Pascal

```
-   -   - ;
Inst1; InstP;   {le corps d'initialisation}
IF condition THEN
      BEGIN
           Instruction i11;
           Instruction i12;
           -   -   -   - ;      } Traitement 1
           -   -   -   - ;
           Instruction i1n;
      END;
                ELSE
      BEGIN
           Instruction i21;
           Instruction i22;
           -   -   -   - ;      } Traitement 2
           -   -   -   - ;
           Instruction i2m;
      END;
-   -   - ;
```

Remarque :

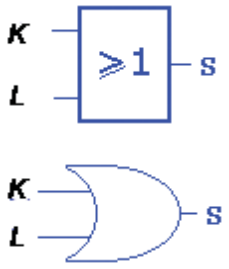
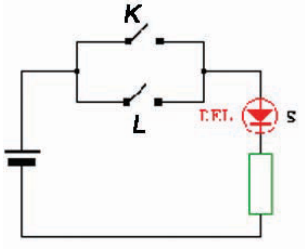
On ne met pas de ";" après le END qui précède le ELSE car la structure conditionnelle n'est pas encore terminée.

Activité 4

Un interrupteur de courant électrique a deux états ON-OFF (1, 0). Selon le montage, un circuit électrique à deux interrupteurs peut réaliser les deux fonctions logiques ET, OU.

Faire une analyse, écrire un algorithme puis la traduction en Pascal du programme intitulé FN_LOGIQUE_OU qui, suivant l'état des interrupteurs K et L, affiche l'état de la lampe LED.

Le principe d'une porte logique OU peut se résumer comme suit : Deux entrées et une sortie. Suivant le niveau logique appliqué aux entrées (niveau 0 ou 1) la sortie sera 0 ou 1.

Fonction	symbole	Table de vérité	Circuit électrique
OU (OR)		Fonction OU (OR)	

Entrée K	Entrée L	Sortie
0	0	0
0	1	1
1	0	1
1	1	1

Analyse

Nom = FN_LOGIQUE_OU

S	L.D.E.	O.U.
4	**Résultat** = Ecrire (LED)	LED K L
3	LED = [ ] **Si** (K=0) ET (L=0) Alors LED ← "Éteinte" **Sinon** LED ← "Allumée" **FinSi**	
2	L = Donnée ("Entrer l'état de l'interrupteur L : ")	
1	K= Donnée ("Entrer l'état de l'interrupteur K : ")	
5	**Fin FN_LOGIQUE_OU**	

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
LED	Chaîne de Caractères	Sauvegarde l'état de la LED
K	Entier	Saisie l'état de l'interrupteur K
L	Entier	Saisie l'état de l'interrupteur L

Algorithme

0) Début FN_LOGIQUE_OU
1) Ecrire ("Entrer l'état de l'interrupteur K: "), Lire (K)
2) Ecrire ("Entrer l'état de l'interrupteur L : "), Lire (L)
3) Si (K=0) ET (L=0) Alors
LED ← " Éteinte "
Sinon
LED ← " Allumée "
FinSi
4) Ecrire (LED)
6) Fin FN_LOGIQUE_OU

Traduction en Pascal

```
PROGRAM FN_LOGIQUE_OU;
USES WINCRT;
VAR
  K,L:INTEGER;
  LED:STRING;

BEGIN
  WRITE ('Entrer l''état de l''interrupteur K : ');READLN(K);
  WRITE ('Entrer l''état de l''interrupteur L : ');READLN(L);
  IF (K=0)AND(L=0) THEN
       LED := 'Éteinte'
     ELSE
       LED := 'Allumée';

  WRITE (LED);
END.
```

II.2.2 Définition

Une structure de contrôle conditionnelle est dite à forme alternative lorsque le traitement dépend d'une condition à deux états : si la condition est évaluée à « VRAI », le premier traitement est exécuté ; si la condition est évaluée à « FAUX », le second traitement est exécuté.

Activité 5

Écrire un programme qui demande à l'utilisateur de saisir 4 valeurs réelles correspondants aux coordonnées de 2 vecteurs $\vec{u}$ (x1 ,y1) et $\vec{v}$ (x2 ,y2) et qui détermine et affiche si ces vecteurs sont orthogonaux entre eux ou pas.

Analyser ce problème et en déduire l'algorithme correspondant.

NB. Deux vecteurs sont orthogonaux si leur produit scalaire est égal à 0.

Pré-analyse

Soient $\vec{u}$ (x1, y1) et $\vec{v}$ (x2 ,y2) dans une base orthonormée de l'ensemble des vecteurs du plan.
u perpendiculaire à **v** équivaut à x1 . x2 + y1 . y2=0.

Analyse

	Nom = ORTHOGONAL	
S	**L.D.E.**	**O.U.**
4	**Résultat** = Ecrire (Nature)	Nature
3	Nature= [ ] **Si** x1*x2 + y1*y2=0 **Alors** Nature←" u est perpendiculaire à v " **Sinon** Nature←" u n'est pas perpendiculaire à v " **FinSi**	x1 x2 y1 y2
2	x2,y2 = Donnée ("Entrer les coordonnés du vecteur v: ")	
1	x1,y1 = Donnée ("Entrer les coordonnés du vecteur u: ")	
5	**Fin ORTHOGONAL**	

Tableau de déclaration des objets

Objet	Type / Nature	Rôle
Nature	Chaîne	Nature des deux vecteurs
x1,x2,y1,y2	Réel	Coordonnées des vecteurs u et v

Algorithme

```
0) Début ORTHOGONAL
1) Ecrire ("Entrer les coordonnés du vecteur u: "), Lire(x1,y1)
2) Ecrire ("Entrer les coordonnés du vecteur v: "), Lire(x2,y2)
3) Si x1*x2 + y1*y2=0
                    Alors
                        Nature←" u est perpendiculaire à v "
                    Sinon
                        Nature←" u n'est  pas perpendiculaire à v "
  FinSi

4) Ecrire (Nature)
5) Fin ORTHOGONAL
```

Activité 6

Traduire l'algorithme ORTHOGONAL en Pascal.

Traduction en Pascal

```
PROGRAM ORTHOGONAL;
USES WINCRT;
VAR
   Nature:STRING;
   X1,y1,X2,Y2:REAL;

BEGIN
  WRITE('Entrer les coordonnés du vecteur u: ');READLN(x1,y1);
  WRITE('Entrer les coordonnés du vecteur u:');READLN(x2,y2);

  IF x1*x2 + y1*y2=0
           THEN
             Nature := 'u est perpendiculaire à v'
           ELSE
             Nature := 'u n''est  pas perpendiculaire à v';

  WRITE (Nature);
END.
```

Retenons

La structure de contrôle conditionnelle simple est utilisée pour définir au maximum deux résultats dépendant de l'évaluation d'une seule condition.
- Une structure de contrôle conditionnelle est dite à forme simple réduite lorsque le traitement dépend d'une condition. Si la condition est évaluée à « VRAI », le traitement est exécuté.
- Une structure de contrôle conditionnelle a une forme alternative si suivant la valeur d'une condition, on exécute soit un traitement 1 soit un traitement 2.

La structure de contrôle conditionnelle simple	Au niveau de l'analyse et de l'algorithme	Au niveau de Pascal
La forme réduite	[Initialisation] **Si** condition **Alors** Traitement **FinSi**	- - -; **If** condition **then** Traitement;

La forme alternative	[Initialisation]
	Si condition **Alors**
	Traitement1
	Sinon
	Traitement2
	FinSi
		- - -;
		If condition **then**
		Traitement1;
		Else
		Traitement 2;

Exercices

Pour chacun des exercices suivants, effectuer une analyse, dresser un algorithme et donner la traduction en Pascal.

Exercice 1

Lire un nombre et afficher sa racine carrée si elle existe.

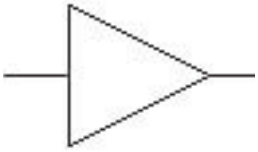
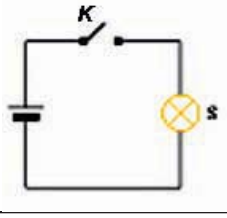
Exercice 2

Demander un caractère à l'utilisateur et tester si ce caractère est majuscule ou minuscule.

Exercice 3

Soit le tableau suivant :

Afficher l'état de la lampe S (allumée ou éteinte) suivant l'état de l'interrupteur K.

<table>
<tr><th>Fonction</th><th>symbole</th><th>Table de vérité</th><th>Circuit électrique</th></tr>
<tr>
<td>OUI</td>
<td>  </td>
<td>
<table>
<tr><th>Entrée K</th><th>Sortie</th></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td></tr>
</table>
</td>
<td> 
</td>
</tr>
</table>

Exercice 4

Ordonner dans le sens décroissant, trois entiers saisis au clavier.

Exercice 5

Saisir une chaîne de caractères et vérifier si elle est composée de plusieurs mots.

Exercice 6

Lire l'année et afficher si elle est éventuellement bissextile.

NB. Si l'année n'est pas divisible par 4, l'année n'est pas bissextile.

Exercice 7

L'université décide d'attribuer un e-mail à ses étudiants selon le modèle suivant : Nom@Campus.net

Il y a cependant une restriction, si le nom excède sept caractères, il doit être tronqué.

Ainsi l'étudiant « MohamedAli » aura comme e-mail: Mohamed@campus.net

Demander le nom de l'étudiant et afficher son e-mail.

Exercice 8

On veut décrire le comportement d'un contrôleur pour une porte d'entrée par carte. Ce contrôleur peut recevoir une indication de lecture d'une carte autorisée (a) ou refusée (r).

Nous avons les contraintes suivantes:

– Si une carte est refusée, il faut sonner immédiatement.
– Lorsqu'une carte est autorisée, la porte s'ouvre.

On se propose de faire un programme intitulé CONTROLEUR qui lit l'état d'un lecteur de carte et affiche le message "Porte en cours d'ouverture" si la carte est autorisée, sinon produit un bip sonore si la carte est refusée.

Exercice 9

Un chef d'entreprise décide d'automatiser les horaires de début de travail, de pause et de fin de travail. Le système consiste à déclancher une alarme et à afficher un message à des heures bien précises de la journée.

On se propose de faire un programme intitulé MINUTERIE qui lit l'heure maintenue par le système d'exploitation et affiche le message "Reprise du travail" et produit deux bips sonores si l'horaire est équivalent à 7 Heures et 30 Minutes ou 14 Heures, sinon affiche le message "Arrêt du travail" et produit un seul bip sonore si l'horaire est équivalent à 12 Heures ou 17 Heures et 15 Minutes.

NB.

- On rappelle que l'écriture de CHR(7) produit un bip sonore.
- La procédure prédéfinei **GETTIME** renvoie l'heure maintenue par le système d'exploitation.

Syntaxe: procedure **GETTIME**(var Heure, Minute, Seconde, Sec100: Word);
Description: les intervalles des valeurs renvoyées sont 0..23 pour heure, 0..59 pour minute, 0..59 pour Seconde et Sec100(centièmes de secondes).

Exercice 10

Saisir un entier de trois chiffres et afficher si cet entier est :

– Cube ou non (la somme des cubes de ses chiffres est égale à lui même).
– Pair - impair
– Positif - négatif

Leçon 2

La structure de contrôle conditionnelle generalisee

Objectifs spécifiques

- Résoudre des problèmes faisant appel à la structure de contrôle conditionnelle généralisée
- Présenter les solutions sous forme d'un algorithme puis d'un programme

Plan de la leçon

I. Introduction

II. Vocabulaire et syntaxe

III. Définition

Retenons

Exercices

Leçon 2

La structure de contrôle conditionnelle generalisee

La logique est l'étude des conditions formelles de la vérité
PIAGET

I. Introduction

Nous avons vu dans la leçon précédente qu'une structure conditionnelle simple conduit à deux alternatives uniquement.
Dans certain problèmes faisant appel à la structure de contrôle conditionnelle, deux situations ne suffisent pas, d'où l'intérêt de la structure de contrôle conditionnelle généralisée qui résout des problèmes dans lesquels plusieurs alternatives sont envisageables.

II. Vocabulaire et syntaxe

Au niveau de l'analyse et de l'algorithme

```
Résultat = [Initialisation ]  Si condition 1 Alors traitement 1
                                Sinon Si condition 2 Alors traitement 2
                                Sinon Si condition 3 Alors traitement 3
                                        - - - - -
                                Sinon Si condition n-1 Alors traitement n-1
                                Sinon traitement n
                              FinSi
```

Remarques :

- Lorsque l'évaluation de la condition 1 produit la valeur :
  VRAI, le traitement 1 sera exécuté puis l'exécution continue avec l'instruction qui suit le Si généralisé.
  FAUX, la condition 2 sera évaluée, si elle produit la valeur :

  VRAI, le traitement 2 sera exécuté puis l'exécution continue avec l'instruction qui suit le Si généralisé.
  FAUX, la condition 3 sera évaluée et ainsi de suite.

- Si aucune des N-1 premières conditions ne produit la valeur VRAI, c'est le traitement N qui sera exécuté.
- Chaque traitement peut comporter une ou plusieurs instructions.
- Si la partition des conditions est exhaustive, il existe une condition et une seule qui est vrai.
- Il est préférable de mettre les événements les plus probables en premier lieu.

Au niveau de Pascal

```
-  -  - ;
{Initialisation}
IF condition 1 THEN traitement 1
     ELSE IF condition 2 THEN traitement 2
     ELSE IF condition 3 THEN traitement 3
          -  -  -
     ELSE IF condition N-1 THEN traitement N-1
     ELSE traitement N ;
-  -  - ;
```

Activité 1

Effectuer une analyse, écrire un algorithme et faire la traduction en Pascal du programme intitulé NATURE_TRIANGLE, qui à partir de trois points donnés A, B et C détermine et affiche la nature du triangle (isocèle en A, isocèle en B, isocèle en C, équilatéral, quelconque).

Pré-analyse

Lorsqu'un triangle ABC est tel que AC = AB (les deux côtés d'extrémité A sont égaux), alors on dit que le triangle est isocèle de sommet A. A est le sommet principal du triangle. Le côté [BC], opposé à A, est appelé base du triangle.

Si un triangle a trois côtés de même longueur, il est dit équilatéral.

Dans le plan cartésien, les points sont définis à l'aide de leurs coordonnées dites cartésiennes.

Soient deux points A et B dans le plan cartésien. On appelle (x_A,y_A) les coordonnées du point A et (x_B,y_B) les coordonnées du point B. La distance AB dans le plan vaut :

$$AB = \sqrt{(x_A - x_A)^2 + (x_A - x_A)^2}$$

Analyse

	Nom = NATURE_TRIANGLE	
S	**L.D.E.**	**O.U.**
8	**Résultat** = Ecrire (Nature)	Nature
7	**Nature= [ ]** **Si** (AC=AB) et (AB=BC) **Alors** Nature←"Triangle équilatéral" **Sinon Si** AB=BC **Alors** Nature←"Triangle isocèle en B" **Sinon Si** AC=BC **Alors** Nature←"Triangle isocèle en C" **Sinon Si** AB= AC **Alors** Nature←"Triangle isocèle en A" **Sinon** Nature←"Triangle quelconque" **FinSi**	AB AC BC
6	BC← RacineCarré (Carré(Xc-Xb)+ Carré(Yc-Yb))	
5	AB← RacineCarré (Carré(Xb-Xa)+ Carré(Yb-Ya))	Xc
4	AC← RacineCarré (Carré(Xc-Xa)+ Carré(Yc-Ya))	Xb
3	Xc,Yc = Donnée ("Entrer les coordonnés du point C: ")	Yc
2	Xa,Yb = Donnée ("Entrer les coordonnés du point B: ")	Yb
1	Xa,Ya = Donnée ("Entrer les coordonnés du point A: ")	Xa Ya
9	**Fin NATURE_TRIANGLE**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
Nature	Chaîne	Sauvegarde la nature du triangle
AB,AC,BC	Réel	Longueur des côtés du triangle
Xc,Xb,Yc,Yb,Xa,Ya	Réel	Saisie les coordonnés des points A, B et C

Algorithme

0) Début NATURE_TRIANGLE
1) Ecrire ("Entrer les coordonnés du point A: "), Lire(Xa,Ya)
2) Ecrire ("Entrer les coordonnés du point B: "), Lire(Xb,Yb)
3) Ecrire ("Entrer les coordonnés du point C: "), Lire(Xc,Yc)
4) AC←RacineCarré (Carré(Xc-Xa)+ Carré(Yc-Ya))
5) AB←RacineCarré (Carré(Xb-Xa)+ Carré(Yb-Ya))
6) BC←RacineCarré (Carré(Xc-Xb)+ Carré(Yc-Yb))

7) Si (AC=AB) et (AB=BC) Alors
Nature←"Triangle équilatéral"
Sinon Si AB=BC Alors
Nature←"Triangle isocèle en B"
Sinon Si AC=BC Alors
Nature←"Triangle isocèle en C"
Sinon Si AB = AC Alors
Nature←"Triangle isocèle en A"
Sinon
Nature←"Triangle quelconque"
FinSi
8) Ecrire (Nature)
9) Fin NATURE_TRIANGLE

Traduction en Pascal

```
PROGRAM NATURE_TRIANGLE;
USES WINCRT;
VAR
  Nature:STRING;
  AC,AB,BC:REAL;
  Xc,Xb,Xa,Yb,Yc,Ya:REAL;

BEGIN
WRITE('Entrer les coordonnés du point A: '); READLN(Xa,Ya);
WRITE('Entrer les coordonnés du point B: '); READLN(Xb,Yb);
WRITE('Entrer les coordonnés du point C: '); READLN(Xc,Yc);
AC:= SQRT(SQR(Xc-Xa)+ SQR(Yc-Ya));
AB:= SQRT(SQR(Xb-Xa)+ SQR(Yb-Ya));
BC:= SQRT(SQR(Xc-Xb)+ SQR(Yc-Yb));
IF (AC=AB)and(AB=BC)THEN
     Nature:='Triangle équilatéral'
ELSE IF AB=BC THEN
     Nature:='Triangle isocèle en B'
ELSE IF AC=BC THEN
     Nature:='Triangle isocèle en c'
ELSE IF AC=AB THEN
     Nature:='Triangle isocèle en A'
ELSE
     Nature:='Triangle quelconque';
WRITE(Nature);
END.
```

III. Définition

Une structure de contrôle conditionnelle est dite généralisée lorsqu'elle permet de résoudre des problèmes comportant plus de deux traitements en fonction des conditions. L'exécution d'un traitement entraîne automatiquement la non exécution des autres traitements.

Activité 2

Dans le but de déterminer la nature d'une solution chimique on utilise un pH_mètre.
Une solution peut être :
- Acide fort si le pH est inférieur à 2
- Acide faible si le pH est entre 2 et 6
- Neutre si le pH est égal à 7
- Base faible si le pH est entre 8 et 12
- Base forte si le pH est supérieur à 13

Pré-analyse

On constate que le pH_mètre ne peut prendre que des valeurs entières, on pourra le considérer comme une variable de type entier.
La solution chimique peut être Acide, Neutre ou Basique.

Analyse

	Nom = SOLUTION	
S	L.D.E.	O.U.
3	**Résultat** = Ecrire (Nature)	Nature pH
2	Nature= [ ] **Si** (pH<2) **Alors** Nature←"Acide fort" **Sinon** Si pH<7 **Alors** Nature←"Acide faible" **Sinon** Si pH=7 **Alors** Nature←"Neutre" **Sinon** Si pH<13 **Alors** Nature←"Base faible" **Sinon** Nature←"Base forte" **FinSi**	
1	pH = Donnée ("Entrer le pH de la solution: ")	
4	**Fin SOLUTION**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
Nature	Chaîne	Sauvegarde de la nature de la solution
pH	Entier	Saisie le pH d'une solution

Algorithme

```
0) Début SOLUTION
1) pH = Donnée ("Entrer le pH de la solution: ")
2) Si pH<2 Alors
                Nature←"Acide fort"
   Sinon  Si pH<7 Alors
                Nature←"Acide faible"
   Sinon  Si pH=7 Alors
                Nature←"Neutre"
   Sinon Si pH<13 Alors
                Nature←"Base faible"
   Sinon
                Nature←"Base forte"
   FinSi
3) Ecrire (Nature)
4) Fin SOLUTION
```

Retenons

La structure de contrôle conditionnelle généralisée offre la possibilité de résoudre des problèmes comportant plus de deux traitements. Une fois qu'une condition est évaluée et qu'un traitement est exécuté les autres ne le seront pas. Cette économie sur les tests se répercute sur le temps d'exécution du programme.

	Au niveau de l'analyse et de l'algorithme	Au niveau de Pascal
La structure de contrôle conditionnelle généralisée	[Initialisation] **Si** cond1 **Alors** trait1 **Sinon Si** cond2 **Alors** trait2 Sinon Si cond3 **Alors** trait3 - - - - - **Sinon Si** cond n-1 **Alors** trait n-1 **Sinon** trait n **FinSi**	- - -; **IF** cond **THEN** trait1 **ELSE IF** cond2 **THEN** trait2 **ELSE IF** cond2 **THEN** trait3 - - - **ELSE IF** cond n-1 **THEN** trait n-1 **ELSE** trait n ;

Exercices

Pour chacun des exercices suivants, effectuer une analyse, écrire un algorithme et faire la traduction en Pascal.

Exercice 1

Lire le nom du port de communication et afficher le débit correspondant.

Port E/S	Débit
Blue tooth	1 Mbits/S
Infrarouge	12 Mbps
IEEE1394	100 - 400 Mbits/s
USB1	12 Mbits/s
USB2	480 Mbits/s

Exercice 2

Écrire un programme qui affiche ce menu :

1. Effectuer une multiplication
2. Effectuer une addition
3. Effectuer une soustraction
4. Effectuer une division

Puis saisit un réel x1, le choix de l'opération et un deuxième réel x2, puis affiche le résultat de l'application de l'operateur choisi sur x1 et x2.
prévoir un message d'erreur au cas où une entrée incorrecte est faite.

Exercice 3

Saisir deux entiers x et y, le premier en décimal et le deuxième en binaire et vérifier si x est supérieur, inférieur ou égal à y puis affiche le résultat.

Exercice 4

Écrire un programme Pascal intitulé **PROTOCOLE** qui lit le nom d'un service internet, puis affiche le protocole correspondant.

Exercice 5

Le conseil scientifique d'une institution effectue un vote pour décider de l'achat de materiels informatiques.
On vous demande d'écrire un programme qui affiche la décision à prendre par le conseil sachant qu'elle est :
« Reportée si le pourcentage des neutress est stictement supérieur à 50%, sinon elle est « acceptée » si le pourcentage des favorables est strictement supérieur à celui des défavorables et « réfusée » dans le cas contraire.

Exercice 6

Lire le type d'un fichier et afficher les extensions possibles.
Exemple :
Type de fichier : Image
Extensions possibles : BMP / TIFF / GIF

Leçon 3

La structure de contrôle conditionnelle à choix

Objectifs spécifiques :

- Résoudre des problèmes faisant appel à la structure de contrôle à choix
- Présenter les solutions sous forme d'un algorithme puis d'un programme Pascal

Plan de la leçon :

Leçon 3

La structure de contrôle conditionnelle à choix

Est nécessaire la proposition qui, étant vraie, n'est pas capable d'être fausse, ou bien celle qui en est capable, mais qui est telle que les circonstances extérieures s'opposent à ce qu'elle soit fausse.

LA PLEIADE

I. Introduction

Si la structure conditionnelle généralisée est une structure à plusieurs traitements en fonction des conditions, la structure conditionnelle à choix est en revanche un choix multiple lorsque plusieurs voies sont possibles selon les différentes valeurs que peut contenir une variable de type scalaire (sélecteur).

Cette structure rend le code source du programme plus lisible.

Une combinaison entre la structure conditionnelle généralisée et la structure conditionnelle à choix multiple est possible.

Activité 1

Pour un entier N donné, lequel de ces nombres est multiple de 5 : N, N +1, N +2, N +3 et N +4.
Effectuer une analyse, écrire un algorithme et faire la traduction en Pascal du programme intitulé MULTIPLE5, qui saisit un entier N et détermine lequel parmi ces nombres est multiple de 5.

Pré-analyse

Pour vérifier que le premier entier N est multiple de cinq, il suffit de montrer que le reste de la division entière de N par 5 vaut 0.
Si le reste est différent de 0 il suffit d'ajouter à N la valeur (5-reste) pour que l'entier (N +5-reste) soit multiple de 5.

Analyse

	Nom = MULTIPLE5	
S	**L.D.E.**	**O.U.**
4	**Résultat** = Ecrire (M)	M
3	M = [ ] **Selon** R **Faire**	R
	0 : M ← "N"	N
	1 : M ← "N +4"	
	2 : M ← "N +3"	
	3 : M ← "N +2"	
	4 : M ← "N +1"	
	Fin Selon	
2	R ← N MOD 5	
1	N = Donnée ("Entrer un entier N : ")	
4	**Fin MULTIPLE5**	

Tableaux de déclaration des objets

Objet	Type/nature	Rôle
M	Chaîne de Caractères	Sauvegarde du résultat
R	Réel	Contient le reste
N	Réel	Saisit de l'entier N

Algorithme

0) Début MULTIPLE5
1) Ecrire ("Entrer un entier N : "), Lire(N)
2) R ← N MOD 5
3) Selon R Faire
0 : M ←"N"
1 : M ←"N +4"
2 : M ←"N +3"
3 : M ←"N +2"
4 : M ←"N +1"
Fin Selon
4) Ecrire (M)
5) Fin MULTIPLE5

Traduction en Pascal

```
PROGRAM MULTIPLE5;
USES WINCRT;
VAR
  N,R : INTEGER;
  M: STRING;
```

```
BEGIN
 WRITE ('Entrer un entire N : '); READLN(N);
 R:=N MOD 5;
 CASE R OF
   0 : M:='N';
   1 : M:='N+4';
   2:  M:='N+3';
   3 : M:='N+2';
   4 : M:='N+1';
 END;
 WRITE (M);
END.
```

II. Vocabulaire et syntaxe

Au niveau de l'analyse et de l'algorithme

[Initialisation] **Selon** sélecteur **Faire**
Valeur 1 : Traitement 1
Valeur 2 : Traitement 2
Valeur 3 : Traitement 3
Valeur 4, Valeur 5, Valeur 6 : Traitement 4
Valeur 7 .. Valeur 10 : Traitement 5

Valeur N : Traitement N
Sinon traitement R
Fin Selon

Commentaire :

- Le sélecteur doit être de type scalaire discret et non réel.
- Plusieurs valeurs séparées par des virgules, auront un même traitement.
- Un intervalle de valeurs peut entraîner un même traitement.
- Un traitement peut comporter une ou plusieurs instructions.
- Si la valeur du sélecteur est différente des valeurs proposées alors le traitement de la clause Sinon sera exécuté.
- La partie **Sinon** est facultative.

Traduction en Pascal

```
-   -   - ;
CASE selecteur OF
     Valeur 1 : Traitement 1;
     Valeur 2 : Traitement 2;
     Valeur 3 : Traitement 3;
     Valeur 4, Valeur 5, Valeur 6 : Traitement 4;
     Valeur 7 .. Valeur 10 : Traitement 5;
     - - -
     Valeur N : Traitement N;
ELSE traitement R;
END;
```

III. Définition

Une structure de contrôle conditionnelle est dite à choix lorsque le traitement dépend de la valeur que prendra le sélecteur. Ce sélecteur doit être de type scalaire (entier ou caractère).

Activité 2

Effectuer une analyse, écrire un algorithme et faire la traduction en Pascal du programme intitulé JOUR, qui saisit une date sous la forme jj/mm/aaaa (chaîne de 10 caractères) où jj représente le jour, mm le mois, aaaa l'année et détermine le jour de la semaine.
Parmi plusieurs possibilités, on peut utiliser la congruence de Zeller dont le résultat est un entier compris entre 0 et 6. (0 signifie samedi, 1 dimanche, 2 lundi, 3 mardi, 4 mercredi, 5 jeudi, et 6 vendredi).
Formule de Zeller :

J= [jj+ ENT(2,6*M-0,2) + (aaaa mod 100) + ENT((aaaa mod 100)/4)) +
ENT((aaaa div 100)/4)) - 2*(aaaa div100)] mod 7.

Où : - jj est le jour du mois.
- M est le mois en utilisant la numération Romaine (1→mars, 2→avril, ... 10→décembre, 11→janvier, ..).
- aaaa représente une année donnée en commençant par 1582.

Exemple : pour la date du 10/11/2006, J =5 alors le jour de la semaine est le vendredi.

Pré-analyse

Formule de Zeller :

J= [jj+ ENT(2,6*M-0,2) + (aaaa mod 100) + ENT((aaaa mod 100)/4)) +
ENT((aaaa div 100)/4)) - 2*(aaaa div100)] mod 7.

Où - ENT désigne la partie entière.
- JJ correspond à un numéro du jour du mois mm de l'année aaaa.
- M représente le rang du mois avec la numération Romaine.
- (aaaa div100) et (aaaa mod 100) sont mis respectivement pour avoir les deux premiers et les deux derniers chiffres de l'année.

Les jours de la semaine sont numérotés selon leur rang, sauf que le samedi correspond à zéro.

Exemple :

Le 20 août 1970 était un jeudi, car J est égal à 96 et le reste de la division par 7 est 5.
Voici le détail des calculs :

J = 20 + [2,6(8 + 1)] + [19/4] - 38 + 70 + [70/4]
J = 20 + 23 + 4 - 38 + 70 + 17 = 96

Si J est négatif, on fait (7n + k) où n est choisi de façon à ce que la valeur de l'expression varie de 0 à 6. Par exemple, si J = -23, on fait 28 - 23 = 5.

Analyse

Nom = JOUR

S	L.D.E.	O.U.
12	**Résultat** = Ecrire (Jour)	Jour J JJ M aaaa mm an E mois day date
11	Jour = [ ] **Selon** J **Faire** 0 : Jour ←"Dimanche" 1 : Jour ←"Lundi" 2 : Jour ←"Mardi" 3 : Jour ←"Mercredi" 4 : Jour ←"Jeudi" 5 : Jour ←"Vendredi" 6 : Jour ←"Samedi" **SiNon** Jour ←"ERREUR" **Fin Selon**	
10	J = [ ] **Si** J < 0 **Alors** J ← -J mod 7 J ← 7 - J **FinSi**	
9	J← (JJ+ TRONC((2.6*M)-0.2)+(aaaa mod 100)+TRONC((aaaa mod 100)/4)+TRONC((aaaa div 100)/4) -2*(aaaa div 100))mod 7	
8	**Selon** mm **Faire** 3 : M ←1 4 : M ←2 5 : M ←3 6 : M ←4 7 : M ←5 8 : M ←6 9 : M ←7 10: M ←8 11: M ←9 12: M ←10 1 : M ←11 2 : M ←12 **Fin Selon**	
7	Valeur(An,aaaa,e)	
6	Valeur(mois,mm,e)	
5	Valeur(day,jj,e)	
4	an ← Sous-chaîne(Date,7,4)	
3	mois ← Sous-chaîne(Date,4,2)	
2	day ← Sous-chaîne(Date,1,2)	
1	date = Donnée ("Entrer une date : ")	
13	**Fin JOUR**	

Tableaux de déclaration des objets

Objet	Type / Nature	Rôle
J	Entier	Calcul du jour par la formule de Zeller
JJ	Entier	Contient le jour
mm	Entier	Contient le mois
aaaa	Entier	Contient l'année
M	Entier	Contient le mois en Romain
E	Entier	Erreur
Jour	Chaîne de caractères	Sauvegarde du jour
date	Chaîne de caractères	Saisie de la date
day	Chaîne de caractères	Contient le jour
mois	Chaîne de caractères	Contient le mois
an	Chaîne de caractères	Contient l'année

Algorithme

```
0) Début JOUR
1) Ecrire ("Entrer une date: "), Lire(Date)
2) day ← Sous-chaîne(Date,1,2)
3) mois ← Sous-chaîne(Date,4,2)
4) an ← Sous-chaîne(Date,7,4)
5) Valeur(day,jj,e)
6) Valeur(mois,mm,e)
7) Valeur(an,aaaa,e)
8) Selon mm Faire
    3 : M ←1
    4 : M ←2
    5 : M ←3
    6 : M ←4
    7 : M ←5
    8 : M ←6
    9 : M ←7
    10: M ←8
    11: M ←9
    12: M ←10
    1 : M ←11
    2 : M ←12
Fin Selon
9) J← (JJ+ TRONC((2.6*M+1)+(aaaa mod 100)+TRONC((aaaa mod
                100)/4)+TRONC((aaaa div 100)/4) -2*(aaaa div 100))mod 7
10) Si J < 0 Alors
                        J ← -J mod 7
                        J ← 7 - J
    FinSi
```

11) Selon J Faire
0 : Jour ←"Dimanche"
1 : Jour ←"Lundi"
2 : Jour ←"Mardi"
3 : Jour ←"Mercredi"
4 : Jour ←"Jeudi"
5 : Jour ←"Vendredi"
6 : Jour ←"Samedi"
SiNon Jour ←"ERREUR"
Fin Selon
12) Ecrire (Jour)
13) Fin JOUR

Traduction en Pascal

```
PROGRAM JOUR;
USES WINCRT;
VAR
 jour,date,day,mois,an:STRING;
 jj,mm,aaaa,M,e,J:INTEGER;
BEGIN
 WRITE('Entrer une date: ');READLN(date);
 day:=COPY(date,1,2);
 mois:= COPY(date,4,2);
 an:= COPY(date,7,4);
 Val(day,jj,e);
 Val(mois,mm,e);
 Val(an,aaaa,e);

 CASE mm OF
       3 : M :=1;
       4 : M :=2;
       5 : M :=3;
       6 : M :=4;
       7 : M :=5;
       8 : M :=6;
       9 : M :=7;
       10: M :=8;
       11: M :=9;
       12: M :=10;
       1 : M :=11;
       2 : M :=12;
 END;
```

```
J:= (JJ+ TRUNC((2.6*M)+1)+(aaaa mod 100)+TRUNC((aaaa MOD
100)/4)+TRUNC((aaaa DIV 100)/4) -2*(aaaa DIV 100))MOD 7;
  If J < 0 THEN
             BEGIN
               J := -J mod 7 ;
               J := 7 - J ;
              END;
CASE J OF
      0 : jour :='Dimanche';
      1 : jour :='Lundi';
      2 : jour :='Mardi';
      3 : jour :='Mercredi';
      4 : jour :='Jeudi';
      5 : jour :='Vendredi';
      6 : jour :='Samedi';
   ELSE jour:='ERREUR';
END;

WRITE(jour);
END.
```

Retenons

La structure conditionnelle à choix permet d'effectuer un choix selon les valeurs possibles d'une expression avec une meilleure lisibilité de la solution.
Cette structure évite l'utilisation d'une longue structure conditionnelle généralisée.
Le choix du traitement dépend de la valeur que prendra le sélecteur.
La valeur du sélecteur doit être de type scalaire.

	Au niveau de l'analyse et de l'algorithme	Au niveau de Pascal
La structure de contrôle conditionnelle généralisée	[Initialisation] **Selon** sélecteur **Faire** Valeur 1 : Traitement 1 Valeur 2 : Traitement 2 Valeur 3 : Traitement 3 Valeur 4, Valeur 5, Valeur 6 : Traitement 4 Valeur 7..Valeur 10 : Traitement 5 - - - Valeur N : Traitement N **Sinon** traitement R **Fin Selon**	- - -; **CASE** sélecteur **OF** Valeur 1 : Traitement 1; Valeur 2 : Traitement 2; Valeur 3 : Traitement 3; Valeur 4, Valeur 5, Valeur 6 : Traitement 4; Valeur 7 .. Valeur 10 : Traitement 5; - - - Valeur N : Traitement N; **ELSE** traitement R; **END;**

Exercices

Pour chacun des exercices suivants, effectuer une analyse, écrire un algorithme et faire la traduction en Pascal.

Exercice 1

Écrire un programme intitulé TOUCHE, qui affiche selon le cas, la nature du caractère (consonne, voyelle, chiffre ou symbole) correspondant à une touche saisie. On considère que le clavier est verrouillé en minuscule.

Exercice 2

Lire un code d'ADN sous forme d'une chaîne de trois caractères, puis déterminer et afficher le code d'ARN correspondant. Sachant que le code d'ADN utilise les lettres A, T, et G et le code d'ARN est obtenu par la correspondance suivante : A→U, T→A, C→G et G→C.

Exercice 3

Saisir le prix unitaire d'un produit et la quantité commandée, affiche le prix à payer, sachant que :

- la remise est de 3% si le montant net est compris entre 1000 D et 3000 D
- la remise est de 5% si le montant net est compris entre 3001 D et 5000 D
- la remise est de 7% si le montant net est supérieur à 5000 D

Exercice 4

Effectuer le cryptage d'un caractère C donnée en utilisant le principe de cryptage suivant :

- si le caractère est une voyelle alors la fonction f(x)=X-1
- si le caractère est une consonne alors la fonction f(x)=X-2
- si le caractère est un chiffre alors la fonction f(x)=X+1
- si le caractère est un symbole alors la fonction f(x)=X+2

X étant le code ASCII du caractère.

Exercice 5

Afficher le jour selon un nombre entré par l'usager (1 = LUNDI, 2 = MARDI, etc.). Votre programme devra détecter les jours erronés.

Exemple :

Entrez un jour : 1
Vous avez choisi LUNDI

Exercice 6

Dans cet exercice, nous nous intéressons aux ordinaux abrégés en anglais, ou le nombre est écrit en chiffres. Les premiers sont :
« 1st », « 2nd », « 3rd », « 4th », etc. (abbreviation de « first, second, third, fourth, ... »).

Pour déterminer le suffixe, on regarde le dernier chiffre du nombre :
- si c'est 1, on ajoute le suffixe est -st
- si c'est 2, le suffixe est -nd
- si c'est 3, le suffixe est -rd
- sinon le suffixe est -th.
- si l'avant-dernier chiffre du nombre est 1, le suffixe est toujours -th.

Écrire un programme qui lit un nombre, et qui affiche l'ordinal anglais abrégée correspondant.

Exercice 7

Écrire un programme intitulé JOURS qui pour une année et un mois donnée affiche le nombre de jours de ce mois.

CHAPITRE

Chapitre 4

Structures de contrôle itératives

Objectifs :

- Utiliser les structures itératives complètes et les structures itératives à condition d'arrêt pour résoudre des problèmes.

- Savoir choisir la structure itérative adéquate pour résoudre un problème.

- Résoudre des problèmes récurrents.

Plan du chapitre :

Leçon 1

Structures de contrôle iteratives complètes

Objectifs spécifiques :

- Utiliser la structure itérative complète pour résoudre des problèmes.
- Résoudre des problèmes récurrents.

Plan de la leçon :

Leçon 1

Les structures de contrôle iteratives complètes

Les concepts sont dits supérieurs s'ils ont sous eux d'autres concepts, qui par rapport à eux sont des concepts inférieurs.

KANT

I. Rappel et définition

Dans la leçon précédente relative aux structures de contrôle conditionnelles, nous avons permis à un programme de prendre des décisions. Nous allons reprendre dans cette leçon, la structure permettant de faire répéter un traitement par l'ordinateur. Cette structure de contrôle s'appelle structure de contrôle itérative complète. Elle est utilisée pour exprimer une répétition d'un traitement un nombre fini de fois.

I.1 Définition

Une structure de contrôle itérative complète exprime la répétition d'un traitement un nombre de fois connu d'avance.

Nous allons voir dans la suite comment exprimer cette structure.

I.2 Vocabulaire et syntaxe

Au cours de l'analyse du problème, l'objet R a une définition itérative complète. On sait aussi que n est le nombre de répétitions. On l'exprime ainsi :

R = [Initialisation] **Pour** c **de** 1 à n **faire**
Traitement
FinPour

La séquence "Traitement" est composée d'instructions ordonnées par ordre d'exécution. On pourra l'exprimer aussi sous la forme :

R = [Initialisation] **Pour** c **de** 1 à n **faire**
Instruction 1
Instruction 2
:
Instruction p
FinPour

On remarque que R est la répétition de ces p instructions. Le nombre de ces répétitions est égal à n. La variable c a servi de compteur qui est automatiquement incrémenté par défaut de 1 par la définition de la structure elle-même.

L'initialisation délimitée par les crochets et placée devant la structure Pour, comporterait les éventuelles définitions nécessaires au bon fonctionnement de la structure itérative. Elle comporte essentiellement les objets, indépendants du compteur de l'itération.

La forme de la structure aura donc la forme suivante :

R = [Inst1, Inst2, …,Instm] **Pour** c **de** 1 à n **faire**
Instruction 1
Instruction 2
.
.
.
Instruction p
FinPour

NB :
1. Le compteur c est de type scalaire (entier, caractères, booléen, etc.).
2. L'incrémentation de c est automatique et fait progresser au successeur de la valeur en cours.

I.3 Traduction en Pascal

```
PROGRAM nom_programme;
USES WINCRT;
VAR   .
      .
      .
BEGIN
      .
      .
      .
   {Instructions d'initialization}
   FOR i:=1 TO n DO
      BEGIN
           .
           Instructions du traitement à répéter
           .
      END;
END.
```

Activité 1

Écrire un programme qui saisit un entier naturel n suivi de n réels à mettre dans un tableau M. Le programme affiche ensuite toutes les valeurs de M supérieurs à 10.

Analyse

	NOM : Supérieurs_à_10	
S	**L.D.E.**	**O.U.**
2 1 3	**Résultat** = sup_10 sup_10 = [ ] **Pour** i **de** 1 **à** n **faire** [ ]**Si** M[i]>10 **Alors** Ecrire (M[i]) **FinSI** **FinPour** M = [**Lire**("n=",n] **Pour** i **de** 1 **à** n **faire** Lire(M[i]) **FinPour** i = compteur **Fin Supérieurs_à_10**	i n M

Tableau de déclaration des objets

Objet	Type/nature	Rôle
i	Entier	Nombre d'itérations
n	Entier	Compteur
M	Tableau de réels	Contient les n réels

Algorithme

0) Début Supérieurs_à_10
1) [Lire("n=",n] Pour i de 1 à n faire
Lire(M[i])
FinPour
2) Pour i de 1 à n faire
[ ]Si M[i]>10 alors
Ecrire (M[i])
FinSI
FinPour

3) Fin Supérieurs_à_10

Traduction en Pascal

```
PROGRAM superieurs_a_10;
USES WINCRT;
VAR  i, n : INTEGER;
     M : ARRAY[1..100] OF REAL;

BEGIN
 WRITE('n=');READLN(n);
 FOR i:=1 TO n DO
        BEGIN
           READLN(M[i]);
        END;
   FOR i:=1 TO n DO
        BEGIN
             IF (M[i]>=10) THEN
                  BEGIN
                     WRITELN(M[i]);
                  END;
        END;
END.
```

I.4 Parcours décroissant

Parfois, on se trouve contraint de faire une itération avec un comptage décroissant. On exprime cette structure comme suit :

R = [Inst1, Inst2, …,Instm] **Pour** c **de** n **à** 1 (pas = -1) **Faire**
Instruction 1
Instruction 2
.
.
.
Instruction p
FinPour

NB : La décrémentation du compteur est automatique en effet, ce dernier passe au prédécesseur de sa valeur en cours. Ici, le compteur décroît de 1.

La traduction en Pascal de cette structure est de la forme :

```
{Instructions d'initialisation}
FOR i:=n DOWNTO 1 DO
     BEGIN
          ...
          Instructions du traitement à répéter
          ...
     END;
```

Remarque :
Généralement, cette exigence provient du fait que le module à répéter utilise les valeurs du compteur.

Activité 2

On se propose de chercher dans un tableau T de noms, les occurrences d'un nom v donné. On demande d'afficher les indices correspondants à ces occurrences d'une façon décroissante.

Analyse

Pour chercher les occurrences de v dans T et les afficher dans un ordre décroissant, il suffit de le parcourir à partir de la fin.

	NOM : Occurrences	
S	**L.D.E.**	**O.U.**
	Résultat = Occur	
3	Occur = [ ] **Pour** i **de** n **à** 1 pas=-1 **faire**	i
	[ ]**Si** T[i]=v **Alors**	n
	Ecrire (i , ",")	T
	FinSI	v
	FinPour	
1	T = [**Lire**("n=",n] **Pour** i **de** 1 à n **faire**	
	Lire(T[i])	
	FinPour	
	i = compteur	
2	v = Donnée ("v = ")	
4	**Fin Occrrences**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
i	Entier	Compteur
n	Entier	Nombre d'itérations
T	Tableau de chaînes de caractères de dimension 1..N_Max	Contient les n noms
N_Max	Constante = 50	Indique la dimension max de T
v	Chaîne de caractères	Le nom à chercher

Algorithme

0) Début Occurrences
1) [Lire("n=",n] Pour i de 1 à n faire
Lire(T[i])
FinPour
2) Lire("v = ",v)
3) Pour i de n à 1 (pas=-1) faire
[ ]Si T[i]=v alors
Ecrire (i, ",")
FinSI
FinPour
4) Fin Occurrences

Traduction en Pascal

```
PROGRAM Occurrences;
USES WINCRT;
VAR  N_Max =50;
VAR  i, n : INTEGER;
        v  : STRING;
T : ARRAY[1..N_Max] OF STRING;

BEGIN
WRITE('n=');READLN(n);
FOR i:=1 TO n DO
 BEGIN
      READLN(T[i]);
 END;
      WRITE('v = ');READLN(v);
  FOR i:=n DOWNTO 1 DO
      BEGIN
         IF (T[i] =v) THEN
               BEGIN
                    WRITELN(i, ',');
               END;
      END;
END.
```

Un cas d'exécution

```
n=6
Bizerte
Tunis
Sousse
Kef
Tunis
Tunis
v = Tunis
6,
5,
2,
```

Cas général :

Il y a des fois où le compteur entre dans le calcul fait par le module à répéter ; en plus les opérations de calcul exigent des valeurs non entières et progressant avec un pas p non entier. L'astuce consiste à chercher par division entière le nombre d'itérations à accomplir et avec une expression généralement linéaire ou affine, revenir au compteur dont on a besoin.

Dans ce cas, la forme générale est :

R = [Inst1, Inst2, …,Instm] **Pour** c de d **à** f (pas = p) **faire**
Instruction 1
Instruction 2
.
.
.
Instruction p
FinPour

NB.
Si p est positif, le parcours est ascendant et si p est négatif, le parcours est descendant.
Le nombre de répétitions est n =1 +E((f - d)/p) et dans ce cas le compteur effectif est c = i*p.
Remarquez que n est toujours positif, c'est le signe de p qui détermine le compteur c.

Traduction en Pascal

```
PROGRAM nom_programme;
USES WINCRT;
VAR
   .
   .
   .

BEGIN
   .
   .
   .
{Instructions d'initialization}
n:=1+ROUND((f-d)/p);
FOR i:=1 TO n DO
        BEGIN
           c:=i*p;
           ...
           Instructions du traitement à répéter
           ...
        END;
END.
```

Activité 3

On se propose de dessiner un cercle par des "*" et pour qu'il soit visible à l'écran, nous allons prendre un rayon égal à R tel que 5<R<12 . Nous rappelons que les équations paramétriques du cercle sont : x(t) = R.cos(t) et y(t) = R.sin(t), t étant un angle mesuré en radians et parcourant l'intervalle [0,2π]. Et pour que ce cercle soit tracé au milieu de l'écran, nous allons procéder à un changement d'origine de la forme suivante : x(t) = 40+R.cos(t) et y(t) = 12+R.sin(t) (l'écran étant de dimension 80x25).
Ecrire un programme qui dessine un cercle avec des "*".

	NOM : Cercle	
S	**L.D.E.**	**O.U.**
	Résultat = cercle	t
2	cercle = [ ] **Pour t de** 0 **à** 2π (pas=p) **faire**	p
	cx=ARRONDI(40+R.COS(t))	cx
	cy=ARRONDI(12+R.SIN(t)	R
	Ecrire(TAB(cx,cy),"*")	cy
	FinPour	
	t = compteur	
	p = 0.01 (Constante)	
1	R = Donnée("Rayon = ")	
3	**Fin cercle**	

Objet	Type/nature	Rôle
t	réel	Angle
cx	Entier	Abscisse du point en cours
cy	Entier	Ordonnée du point en cours
p	Constante réelle = 0.01	Le pas angulaire
pi	Constante réelle = 3.14	Valeur approchée à 0.01 près de π
R	Réel	Le rayon du cercle à tracer

Nous remarquons dans l'analyse faite que le compteur est réel ; ce genre de compteur ne convient pas avec la définition que nous avons présentée précédemment. Nous allons remédier à cette défaillance en calculant d'abord le nombre d'itérations et en utilisant t comme angle auquel on ajoute p à chaque itération.
Le nombre d'itérations est E(2π/p)+1.
De l'analyse précédente, nous obtenons l'algorithme suivant en tenant compte de la légère modification sur le compteur.

Algorithme

0) Début cercle
1) lire(R)
*2) [n←1+ARRONDI(2*pi/p), t← -p] Pour i de 1 à n faire*
t←t+p
cx← ARRONDI((40+R.COS(t))
cy← ARRONDI((12+R.SIN(t))
Ecrire(TAB(cx,cy),"")*
FinPour
3) Fin cercle

Traduction en Pascal

```
PROGRAM cercle;
USES WINCRT;
   CONST pi =3.14;
         p = 0.01;
   VAR   i, n, cx, cy: INTEGER;
         t,R:REAL;
BEGIN
 READLN(R);
 n := 1+ROUND(2*pi/p);
 t:= -p;
 FOR i:=1 TO n DO
   BEGIN
     t:=t+p;
     cx:=40+ ROUND(R*COS(t));
     cy:=12+ ROUND(R*SIN(t));
     GOTOXY(cx, cy);
     WRITE("*");
    END;
END.
```

Exercice corrigé 1

Ecrire un programme qui saisit un mot composé de lettres en majuscules et remplace chacune de ses lettres d'ordre pair par sa symétrique dans la liste de l'alphabet par rapport au milieu. Par exemple, la lettre A sera remplacée par la lettre Z et réciproquement, la lettre J par la lettre Q et réciproquement.
A la fin, le programme affiche la transformée de cette chaîne.

Analyse

Nous tenons à remarquer qu'une chaîne de caractères est vue comme un tableau de caractères. Nous allons exploiter cette propriété dans la résolution du problème.

	NOM : mot_transformé	
S	**L.D.E.**	**O.U.**
2 1 3	**Résultat** = Ecrire("Mot transformé : ",mot_t) mot_t = [Lire("mot = ", mot), mot_t←mot, L ←LONG(mot)] **Pour** c **de** 2 **à** l (pas=2) **faire** code ←ORD(mot[c]) mot_t[c] ← CHR(2*x0 - code +1) **FinPour** **Fin mot_transformé**	mot_t mot L c x0 code

Remarques :

- le mot transformé mot_t est initialisé à mot car il y a que les caractères pairs qui seront modifiés.
- Comme le compteur est automatiquement incrémenté de 1, nous allons utilser un autre compteur alant de 1 à L DIV 2 et c sera égal à 2i.

Tableau de déclaration des objets

Objet	Type/nature	Rôle
mot_t	Chaîne de caractères	Le mot transformé
mot	Chaîne de caractères Entier	Le mot initial
L	Entier	La longueur du mot
c	Entier	Compteur
i	Entier	Compteur
code	Entier	Code du caractère en cours
x0	Constante x0=77	Ordre ASCII de "M" milieu de "A".."Z"

Algorithme

0) Début mot_transformé
1) [Lire("mot = ", mot), mot_t←mot, L ←LONG(mot)] Pour i de 1 à L DIV 2 faire
c←2.i
code← mot[c]
*mot_t[c]← CHR(2*x0 - code+1)*
FinPour
2) Ecrire("Mot transformé : ",mot_t)
3) Fin mot_transformé

Traduction en Pascal

```
PROGRAM mot_transformé;
USES WINCRT;
   CONST x0=77 ;
   VAR i, L, c, code : INTEGER;
        mot,mot_t :STRING;

BEGIN
WRITE('mot = '); READLN(mot);
Mot_t:=mot;
L:= LENGTH(mot);
FOR i:=1 TO (L DIV 2) DO
        BEGIN
          c:=2*i;
          code:=mot[c];
          mot_t[c]:=CHR(2*x0-code+1);
        END;
WRITELN('Mot transformé : ',mot_t);
END.
```

Un cas d'exécution

```
mot = INFORMATIQUE
Mot transformé : IMFLRNAGIJUV
```

II. Les itérations complètes récurrentes

Souvent, certains résultats sont définis d'une façon récurrente. Dans de tels cas, le résultat se forme au fur et à mesure et à une étape donnée, il dépend d'un certain nombre de résultats précédents. Une telle structure est dite itérative récurrente. En fait, les résultats intermédiaires sont gérés par une relation de récurrence. Si cette relation lie deux éléments successifs, on dit que la récurrence est d'ordre 1, si elle lie trois éléments successifs, on dit qu'elle est d'ordre 2, etc.

Activité 13

Ecrire un programme qui calcule la factorielle d'un entier naturel n et l'affiche à l'écran.

Analyse

Remarquons que le calcul de la factorielle a une définition itérative complète et à l'étape i, la factorielle est égale à sa valeur à l'étape i-1 multipliée par i.

	NOM : Factorielle	
S	**L.D.E.**	**O.U.**
3 2 1 4	**Résultat** = Ecrire (n,"! = ", fact) fact = [fact ← 1] **Pour** i **de** 1 **à** n **faire** fact ← fact * i **FinPour** n = DONNEE("n = ") **Fin Factorielle**	n i fact

Objet	Type/nature	Rôle
n	Entier	Le nombre dont on veut calculer la factorielle
fact	Entier	la valeur de n!
i	Entier	Un compteur

Algorithme

0) Début Factorielle
1) Lire ("n = ", n)
2) [fact←1] Pour i de 1 à n Faire
*fact ← fact * I*
FinPour
5) Ecrire(n,"! = ",fact)
6) Fin Factorielle

Traduction en Pascal

```
PROGRAM Factorielle;
USES WINCRT;
VAR n, I, fact :INTEGER;

BEGIN
  WRITE('n = '); READLN(n);
  fact := 1;
  FOR I:=1 TO n DO
      BEGIN
        fact :=fact*I;
      END;
  WRITELN(n,'! = ',fact);
END.
```

Un cas d'exécution

```
n = 5
5! = 120
```

Exercice corrigé 2

n individus passent un examen composé de 3 épreuves de coefficients respectifs 2, 2 et 4. Après la correction, on se propose de faire le calcul des moyennes, la moyenne générale par épreuve, la moyenne générale de toutes les épreuves pondérées et le nombre d'admis sachant qu'un candidat est déclaré admis s'il obtient une moyenne supérieure ou égale à 10 .

Analyse

Pour résoudre ce problème, nous allons choisir quatre tableaux, un pour chacune des trois épreuves (N1, N2, N3) et un dernier pour la moyenne (M).

	NOM : Les_admis	
S	**L.D.E.**	**O.U.**
5	**Résultat** = (Ecrire ("Nombre d'admis : ", n_ad), "Moyenne Générale = ", MG)	n_ad MG
4	n_ad = [n_ad ← 0] **Pour** i **de** 1 **à** n **faire** [ ] **Si** (M[i]e10) **alors** n_ad ← n_ad +1 **FinSi** **FinPour**	i n M
3	MG ← TOT_M / n	
2	Tot_M = [Tot_M ← 0] **Pour** i **de** 1 **à** n **faire** Tot_M ← Tot_M +M[i] **FinPour**	TOT_M N1 N2
1	M = [Lire("n=",n)] **Pour** i **de** 1 à n **faire** Lire("Note 1 =",N1[i]) Lire("Note 2 =",N2[i]) Lire("Note 3 =",N3[i]) M[i] ← (2 * N1[i]+2 * N2[i]+4 * N3[i])/8 **FinPour**	N3
6	**Fin Les_admis**	

Tableau de déclaration d'un nouveau type

Type
Vecteur : tableau [1..50] de réels

Objet	Type/nature	Rôle
n_ad	Entier	Nombre d'admis
MG	Réel	Moyenne générale
i	Entier	Compteur
n	Entier	Nombre de candidats
M	Vecteur	Tableau comportant les moyennes des cadidats
TOT_M	Réel	Le total des moyennes
N1	Vecteur	Le tableau de la 1ère note
N2	Vecteur	Le tableau de la 2ème note
N3	Vecteur	Le tableau de la 3ème note

Algorithme

0) Début Les_admis
1) Lire ("n = ", n) Pour i de 1 à n faire
Lire("Note 1 =",N1[i])
Lire("Note 2 =",N2[i])
Lire("Note 3 =",N3[i])
*M[i] ← (2 * N1[i]+2 * N2[i]+4 * N3[i])/8*
FinPour
2) [Tot_M←0] Pour i de 1 à n faire
Tot_M ← Tot_M +M[i]
FinPour
3) MG ←TOT_M/n
4) [n_ad ← 0] Pour i de 1 à n faire
Si (M[i]e10) alors
n_ad ← n_ad +1
FinSi
FinPour
5) Ecrire ("Nombre d'admis : ", n_ad), "Moyenne Générale = ", MG)
6) Fin Les_admis

Traduction en Pascal

```
PROGRAM Les_admis;
USES WINCRT;
TYPE vecteur = ARRAY[1..50] OF REAL;
VAR n_ad, n, i, fact :INTEGER;
MG, TOT_M : REAL;
N1, N2, N3, M  : vecteur;
BEGIN
   WRITE('n = ');(READLN(n);
      FOR i:=1 TO n DO
```

```
    BEGIN
         WRITE('Note 1 = ');READLN(N1[i]);
         WRITE('Note 2 = ');READLN(N2[i]);
         WRITE('Note 3 = ');READLN(N3[i]);
         M[i]:=(2* N1[i]+2* N2[i]+4*N3[i])/8;
       END;
    TOT_M:=0;
    FOR i:=1 TO n DO
         BEGIN
             TOT_M:=TOT_M+M[i];
         END;
      MG := TOT_M / n;
      n_ad:=0;
    FOR i:=1 TO n DO
         BEGIN
              IF (M[i]>=10) THEN
                BEGIN
                   n_ad:=n_ad+1;
                END;
         END;
    WRITELN('Nombre d''admis : ',n_ad),'Moyenne générale = ',MG);
END.
```

Un cas d'exécution

```
n = 3
Note 1 = 12
Note 2 = 13
Note 3 = 14
Note 1 = 6
Note 2 = 8
Note 3 = 9
Note 1 = 11
Note 2 = 15
Note 3 = 18
Nombre d'admis : 2   Moyenne générale = 12.25
```

Retenons

- Une itération complète consiste en la répétition d'un traitement un nombre de fois connu d'avance.
- Dans une itération complète, le compteur est incrémenté ou décrémenté automatiquement.
- Une itération est dite récurrente si le traitement à l'étape i dépend des étapes qui le précèdent. En général, les résultats intermédiaires sont gérés par une relation de récurrence. En fait, on s'intéresse au résultat final de l'itération.

Exercices

Exercice 1

Écrire un programme qui défile le nom et le prénom de l'utilisateur de la droite vers la gauche sur la ligne du milieu de l'écran.

Exercice 2

Écrire un programme qui fait tourner des étoiles sur les quatre bords de l'écran. C'est en fait une simulation d'un jeu de lumière

Exercice 3

Écrire un programme qui dessine sur deux lignes de l'écran, deux caractères "*" sachant que l'un se déplace de la gauche vers la droite et l'autre se déplace de la droite vers la gauche.

Exercice 4

Reprendre le programme de l'exercice 1 et transformer le défilement de telle façon que la chaîne formée du nom et du prénom rentre par la droite et sort par la gauche. Temporiser ce défilement en provoquant un ralentissement sensible.

Exercice 5

Écrire un programme qui permet de trouver puis d'afficher tous les entiers à deux chiffres ayant le chiffre des unités divisible par le chiffre des dizaines.
Exemple : 24 vérifie cette propriété, en effet, 2 divise 4.

Exercice 6

Écrire un programme qui affiche tous les entiers positifs de deux chiffres tels que la somme de ses deux chiffres divise leur produit.

Exercice 7

Écrire un programme qui saisit un entier naturel n non nul et supérieur à 20 puis remplit un tableau R par n réels aléatoires compris entre 0 et 20 en utilisant la fonction HAZARD préprogrammée. Ensuite, Il calcule la moyenne arithmétique de ces n réels et affiche les éléments de R supérieurs ou égaux à MG ainsi que leur nombre, il en fait de même pour les éléments de R inférieurs à MG.

Exercice 8

Écrire un programme Pascal intitulé **OCCURENCE** qui permet de saisir une chaîne de caractères CH puis d'afficher les occurrences des voyelles qui figurent dans CH.

Exemple :
Si CH ='LYCEE 25 juillet'
Le programme OCCURENCE affichera les résultats suivants :

L'occurence de 'E' est 3
L'occurence de 'Y' est 1
L'occurence de 'U' est 1
L'occurence de 'I' est 1

Remarque : la recherche de l'occurrence ne fait pas de distinction entre les voyelles majuscules et minuscules.

Exercice 9

Écrire un programme qui saisit un texte en français et détermine les fréquences des voyelles et les met dans un tableau de 6 éléments. Le programme affiche ensuite la fréquence de chacune des voyelles.
Le texte pourra comporter des lettres en majuscules ou en minuscules ainsi que les caractères accentués.

Exercice 10

On dispose de deux tableaux T1 et T2 contenant respectivement n et m entiers positifs et non nuls.
On désire chercher dans T2 tous les diviseurs d'un élément donné de T1.

Exemple :

T1	23	15	10	277	300	34
	1	2	3	4	5	6
T2	3	6	5	1		

Si indice = 2 alors 3, 5 et 1 seront affichés à l'écran.

Écrire un programme Pascal qui permet de saisir les deux tableaux T1 et T2 et l'indice d'un élément p de T1 puis d'afficher à l'écran tous les diviseurs de p figurant dans T2.

Exercice 11

Soit un tableau T1 de n éléments ($1 \leq n \leq 100$) . Les éléments de T1 sont des entiers naturels de trois chiffres.
On se propose de remplir un tableau T2 de la façon suivante :
T2[i] est égal à la somme des carrés des chiffres de T1[i].

Exemple :

Si T1[i] = 254 alors T2[i] = 22 + 52 + 42 = 45

Écrire un programme Pascal qui permet de saisir les éléments de T1, de remplir puis d'afficher le tableau T2.

Exercice 12

Écrire un programme Pascal qui saisit un tableau A de n chaînes de caractères, cherche et affiche la longueur de la chaîne la plus longue puis toutes les chaînes ayant cette longueur.

Leçon 2

Structures de contrôle iteratives à conditions d'arrêt

Objectifs spécifiques:

- Utiliser la structure itérative à condition d'arrêt pour résoudre des problèmes.
- Résoudre des problèmes récurrents basés sur des itérations à condition d'arrêt.

Plan de la leçon

Leçon 2

Les structures de contrôle iteratives à condition d'arrêt

Ce n'est pas assez de faire des pas qui doivent un jour conduire au but, chaque pas doit être lui même un but en même temps qu'il nous porte en avant.

GOETHE

I. Introduction

Nous revenons dans cette leçon à une structure de contrôle permettant à l'ordinateur de répéter un traitement donné. Nous avons vu le cas des répétitions dont le nombre est connu d'avance. Nous l'avons appelé : structure itérative complète. Cependant, plusieurs problèmes nécessitent les répétitions d'un traitement dont on ne connaît pas le nombre. C'est une condition qui doit gérer l'arrêt des répétitions.
Nous allons vous présenter des activités auxquelles vous êtes relativement familiarisées.

Activité 1

Une personne en possession d'une carte interbancaire (CIB) se présente à un distributeur pour retirer de l'argent. Cette personne a des doutes concernant son code secret. Le système ne lui permet que trois essais au maximum.
Ecrire les étapes du programme permettant la lecture du code secret.

Réponse :

En effet, dès que la personne aura mis sa carte dans le distributeur, il va entrer dans une boucle qui va lui permettre la saisie de son code. Si la première saisie est fausse, le distributeur va lui donner une seconde chance. Si après cette chance, le code saisi est encore erroné, le distributeur donne la troisième et la dernière chance, après quoi, la carte sera confisquée si le code n'est pas encore correct.
Nous pouvons formuler cette action par :

```
[Entrer la carte] Répéter
                      Incrémenter l'essai
                      Saisir le code
                  Jusqu'à (code correct) ou (essai=3)
```

Activité 2

Vous êtes dans une station de transport public à attendre le moyen de transport désiré. Ce dernier arrive, le convoyeur procède à faire monter les voyageurs avec la condition qu'aucun ne doit voyager debout.
Écrivez l'action faite par le convoyeur.

Réponse :
Dans ce cas et en arrivant à une station, le convoyeur avertit le chauffeur d'ouvrir les portes s'il reste des places vacantes et commence l'action itérative suivante. Tant qu'il y a encore des places, faire monter les voyageurs.
Cette opération pourra être formulée comme suit :

```
[Arrêter le bus] Tant Que (il y a des places vacantes) faire
                     Monter voyageur
                 Fin
```

Vous avez vu que dans chacune des deux activités, nous avons utilisé une structure itérative particulière. Ce sont les deux formulations traduisant la structure itérative à condition d'arrêt.

Définition

On appelle structure de contrôle itérative à condition d'arrêt l'action qui consiste à répéter un traitement donné et que l'arrêt est géré par une condition.
Il existe deux formulations pour traduire une telle structure :
La structure Répéter …. Jusqu'à …
et la structure : Tant Que … Répéter ….

II. La structure Répéter … Jusqu'à …

II.1. Vocabulaire et syntaxe

Les expressions que nous allons présenter sont utilisées dans l'analyse et dans l'algorithme.

Formulation 1 : La structure Répéter … Jusqu'à …
On suppose que R est un objet dont la définition est itérative à condition d'arrêt et que la formulation 1 s'adapte le mieux, nous traduisons ce fait par :

```
R = [Initialisation ]   Répéter
                            Traitement
                        Jusqu'à (Condition darrêt)
```

Remarques :

- La définition de R commence par une initialisation éventuelle. Elle comporte les définitions nécessaires à la marche du processus d'itérations.
- "Traitement" est le jeu d'instructions à répéter.
- "Condition d'arrêt" est une expression booléenne supervisée directement ou indirectement par le traitement répété pour la faire passer à l'état vraie afin que l'itération s'arrête.

Nous pouvons écrire plus en détail cette formulation :

R = [Inst1, Inst2, …,Instm] **répéter**
Instruction 1
Instruction 2
.
.
.
Instruction p
Jusqu'à (Condition d'arrêt)

II.2. Traduction en Pascal

```
PROGRAM nom_programme;
USES WINCRT;
VAR   .
      .
      .
BEGIN
  .
  .
  .
  {Instructions d'initialisation}
  REPEAT
      ...
      Instructions du traitement à répéter
      ...
  UNTIL (Condition d'arrêt);
  ...
END.
```

Activité 3

Écrire un programme qui cherche si une valeur v existe dans un tableau B. B est composé de n réels. v est une valeur réelle donnée. Le programme affichera un message de la forme :
"v existe dans B"
ou "v n'existe pas dans B".

Analyse

Nous allons procéder à un parcours du tableau B en affectant une variable booléenne qu'on nommera v_y_est. Cette itération à condition d'arrêt s'arrête soit quand on aura trouvé v soit qu'on aura fini de parcourir tout le tableau B.

	NOM : Recherche	
S	**L.D.E.**	**O.U.**
4	**Résultat = Écrire**(v, décision, " dans le tableau")	décision
3	décision = [décision ← " n'existe pas"] **Si** v_y_est **alors**	v_y_est
	décision ← " existe"	
	FinSI	
2	v_y_est = [Lire("v = ", v),i← 0] **Répéter**	i
	i ← i+1	B
	v_y_est ← (B[i]=v)	v
	Jusqu'à (v_y_est) OR (i=n)	n
1	B = [**Lire**("n = ", n)] **Pour** i **de** 1 à n **faire**	
	Lire(B[i])	
	FinPour	
5	**Fin Recherche**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
décision	Chaîne de caractères	Comporte la bonne expression
v_y_est	Booléen	Variable logique passant à vrai si v est dans B
v	Réel	Nombre à chercher dans le tableau
i	Entier	Compteur
B	Tableau de réels	Tableau comportant les n réels
n	Entier	Le nombre d'éléments de B

Algorithme

0) Début Recherche
1) [Lire("n=",n] Pour i de 1 à n faire
Lire(B[i])
FinPour

2) [Lire("v = ",v),i← 0] Répéter
i ← i+1
v_y_est ← (B[i]=v)
Jusqu'à (v_y_est) OR (i=n)
3) [décision ← " n'existe pas"] Si v_y_est alors
décision ← " existe"
FinSI
4) Écrire (v, décision, " dans le tableau")
5) Fin Recherche

Traduction en Pascal

```
PROGRAM Recherche;
USES WINCRT;
VAR i, n : INTEGER;
     decision : STRING;
     v_y_est : BOOLEAN;
     v : REAL;
     B : ARRAY[1..100] OF REAL;

BEGIN
  WRITE('n=');READLN(n);
  FOR i:=1 TO n DO
        BEGIN
             READLN(B[i]);
        END;
  WRITE('v = ');READLN(v);
  i:=0;
  REPEAT
    i:=i+1;
    v_y_est:=(B[i]=v)
  UNTIL (v_y_est) OR (i=n);

  decision :='n''existe pas';

  IF v_y_est  THEN
    BEGIN
      decision:=  'existe';
    END;
   WRITELN(v:5:3,'  ', decision, ' dans le tableau');
END.
```

Un cas d'exécution

```
n=5
1
5.89
6
44
124.10
v = 44
44.000  existe dans le tableau
```

Nous allons voir dans l'activité qui suit une application très fréquente de la formulation "Répéter … Jusqu'à …". En effet, elle s'adapte le mieux dans les cas de saisie de données assujetties à des conditions de contrôle.

Activité 4

Écrire un programme qui cherche la dernière position d'une valeur v dans un tableau B de n réels. L'entier n est compris entre 20 et 30. Tous les réels de B ainsi que v sont dans l'intervalle [0,20].

Analyse

Le problème ressemble étroitement à celui de l'activité 3 seulement les données sont astreintes à des conditions ; en plus, la recherche de la dernière valeur de v dans B nous oblige à faire le parcours de B en commençant par la fin pour éviter un parcours total.

	NOM : cherche_dernier	
S	**L.D.E.**	**O.U.**
6	**Résultat = Écrire** (v, décision, " dans le tableau")	décision
5	décision = [décision ← " n'existe pas"] **Si** dp≠0 **alors**	dp
	STR(dp,s)	s
	décision ← " a une dernière place " + s + " "	
	FinSI	i
4	dp = [i← n+1, dp ← 0] **Répéter**	n
	[i ← i-1] **Si** (B[i]=v) **alors**	
	Dp ← i	B
	Jusqu'à (dp≠0) OR (i=1)	
3	v = [ ] **Repéter**	
	Lire("v = ", v)	
	Jusqu'à (v>=0) et (v<=20)	
2	B = [ ] **Pour** i **de** 1 **à** n **faire**	
	[ ]**Répéter**	
	Lire(B[i])	
	Jusqu'à (B[i]>=0) et (B[i]<=20)	
	FinPour	
1	n = [ ] **Repéter**	
	Lire("n = ", n)	
	Jusqu'à n dans [20..30]	
7	**Fin cherche_dernier**	

Table de déclaration des objets

Objet	Type/nature	Rôle
décision	Chaîne de caractères	Comporte la bonne expression
dp	Entier	Dernière position de v dans B
s	Chaîne de caractères	Conversion de dp en chaîne
i	Entier	Compteur
n	Entier	Le nombre d'éléments de B
B	Tableau de réels	Tableau comportant les n réels

Algorithme

```
0) Début cherche_dernier
1) Répéter
        Lire("n=",n)
   Jusqu'à n dans [20..30]
 2) Pour i de 1 à n faire
        Répéter
              Lire(B[i])
        Jusqu'à (B[i]>=0) et (B[i]<=20)
   FinPour
3) Répéter
        Lire("v=",v)
   Jusqu'à (v>=0) et (v<=20)
4) [i←n+1, dp ← 0] Répéter
                        [i ← i-1] Si  (B[i]=v) alors
                                    dp← i
                                FinSi
                    Jusqu'à (dp≠0) OR (i=1)
5)   [décision ← " n'existe pas"] Si dp≠0  alors
                                    STR(dp,s)
                                    décision ← " a une dernière place  " + s +" "
                                FinSI
6) Écrire (v, décision, " dans le tableau")
7) Fin cherche_dernier
```

Traduction en Pascal

```
PROGRAM Recherche;
USES WINCRT;
VAR  i, n,dp : INTEGER;
     decision,s : STRING;
     v : REAL;
     B : ARRAY[1..100] OF REAL;
```

```
BEGIN

  REPEAT
     WRITE('n=');READLN(n);
  UNTIL n IN [20..30];

  FOR i:=1 TO n DO
     BEGIN
       REPEAT
       WRITE('   ');READ(B[i]);
       UNTIL (B[i]>=0 )AND(B[i]<=20);
     END;

  REPEAT
     WRITE('v=');READLN(v);
  UNTIL (v>=0 )AND(v<=20);
   dp:=0;
   i:=n+1;

  REPEAT
   i:=i-1;
     IF B[i]=v THEN
                dp:=i;
  UNTIL(dp<>0) OR (i=1);

   decision:=' n''existe pas';

  IF dp<>0  THEN
     BEGIN
       STR(dp,s);
       decision:=' a une dernière place ' +s+ ' ';
     END;

  WRITELN(v:5:3,'  ',decision, ' dans le tableau');
END.
```

Un cas d'exécution

```
n=20
  7
  6
  4
  8
  12
  15
  17
  5
  20
  14
  12
  9
  7
  6
  4
  3
  4
  2
  11
  13
v=4
4.000   a une dernière place 17  dans le tableau
```

Commentaires :

Vous avez certainement remarqué la différence entre les solutions des deux activités. En effet, pour la recherche de v dans le tableau B, nous avons utilisé une variable booléenne dans la solution du 1er problème, cette option n'est plus adéquate pour le second, nous avons besoin d'une donnée précise en l'occurrence la dernière position de v dans le tableau B.
Nous avons utilisé une fonction de la bibliothèque STR et une expression de concaténation que nous avons formulé par l'opérateur additif +. Bien sûr, cette formulation n'est pas unique, nous pouvons garder l'affichage de dp comme un entier.

II.3. Les problèmes récurrents

Nous avons vu comment sont définis les problèmes récurrents et comment les résoudre avec une itération complète. Il en est de même pour les itérations à condition d'arrêt. En effet, le résultat final se forme au fur et à mesure jusqu'à ce qu'une condition d'arrêt devienne vraie. Dans l'activité suivante, nous vous proposons un problème classique où nous allons utiliser la structure itérative à condition d'arrêt dans un problème récurrent.

Activité 5

On se propose de chercher et d'afficher la moyenne arithmétique d'une suite de réels. A priori, on ne connaît pas le nombre d'éléments de cette liste, par contre, on sait que sa fin sera marquée par le nombre négatif -1. Ce dernier n'entre pas dans le calcul de la moyenne. Le caractère récurrent réside bien entendu dans le calcul de la moyenne.

Analyse

	NOM : Moyenne	
S	**L.D.E.**	**O.U.**
3	**Résultat = Écrire** ("Moyenne = ", mg)	mg
2	mg = total/nbre	total
1	(total, nbre) = [total← 0, nbre← 0] **Répéter**	nbre
	[Lire("x= ",x)]**Si** (x>0) alors	x
	total ← total + x	
	nbre ← nbre +1	
	FinSi	
	Jusqu'à (x= -1)	
4	**Fin Moyenne**	

Table de déclaration des objets

Objet	Type/nature	Rôle
mg	Réel	La moyenne arithmétique des réels
total	Réel	Le total des réels
nbre	Entier	Le nombre de réels positifs
x	Réel	Variable servant à la saisie des réels

Algorithme

0) Début Moyenne
1) [total← 0, nbre← 0] Répéter
[Lire("x= ",x)]Si (x>0) alors
total ← total + x
nbre ← nbre +1
FinSi
Jusqu'à (x=-1)
2) mg = total/nbre
3) Écrire ("Moyenne = ", mg)
4) Fin Moyenne

Traduction en Pascal

```
PROGRAM Moyenne;
USES WINCRT;
VAR     nbre : INTEGER;
        mg, total, x : REAL;

BEGIN
  total:=0;
  nbre:=0;
  REPEAT
     WRITE('x=');READLN(x);
     IF (x>0) THEN
        BEGIN
          total:=total + x;
          nbre:=nbre + 1
        END;
  UNTIL (x=-1);
        mg:=total/nbre;
  WRITE('Moyenne = ',mg:2:);
END.
```

Un cas d'exécution

```
x=3
x=5
x=7
x=-87
x=7
x=-1
Moyenne = 5.50
```

III. La structure Tant Que … Faire …

III.1 Vocabulaire et syntaxe

On suppose que l'objet R a une structure itérative à condition d'arrêt et que nous avons opté pour la formulation "Tant Que … Faire…"

R = [Initialisation] **Tant Que** Not(Arrêt) **Faire**
Traitement
FinTantQue

Remarques :

- "Traitement" est le jeu d'instructions à répéter.
- "Not(Arrêt)" est une expression booléenne supervisée directement ou indirectement par le traitement répété pour la faire passer à l'état vrai afin que l'itération s'arrête.
- "FinTantQue" sert à délimiter les instructions qui vont former "Traitement". Remarquons que pour la 1ère formulation "Répéter …jusqu'à …", la délimitation de "Traitement" est faite naturellement par les deux termes de la structure "Répéter" et "Jusqu'à"
- La partie "Initialisation" comportera les éventuelles définitions qui serviront au processus itératif.

Remarque importante :

Dans les deux formulations, il faut faire attention aux deux cas d'entrée et de sortie de la boucle et de faire les bonnes initialisations et les bons emplacements des instructions relatives à l'avancement dans l'itération. Ceci est souvent une source d'erreurs.

Nous pouvons écrire plus en détail cette formulation :

R = [Inst1, Inst2, …,Instm] **Tant Que** Not(arrêt) **Faire**
Instruction 1
Instruction 2
.
.
.
Instruction p
FinTantQue

III.2 Traduction en Pascal

```
PROGRAM nom_programme;
USES WINCRT;
VAR      .
         .
         .
BEGIN
  .
  .
  .
  {Instructions d'initialisation}
  WHILE Condition DO
   BEGIN
        ...
        Instructions du traitement à répéter
        ...
   END;
   .
   .
   .
END.
```

Activité 6

On se propose de déterminer le PGCD de deux entiers naturels m et n.
Le principe de la recherche repose sur les propriétés suivantes :
le PGCD de m et de 0 est m et tout diviseur de m et n est aussi diviseur du reste de m par n.

Donc on va pouvoir diminuer progressivement les valeurs des couples m et n jusqu'à arriver à annuler l'un des termes. Remarquons que cette méthode de reste converge rapidement vers le résultat c'est-à-dire que si on garde m plus grand que n, le PGCD des deux entiers sera égal à m dès que n devient 0.

Remarque :
De la propriété "tout diviseur de m et n " divise leur différence et dont on pourra faire facilement une démonstration comme pour le reste, nous déduisons un autre procédé de calcul de PGCD en utilisant la différence. Généralement, la méthode de la différence converge vers 0 plus lentement que la méthode du reste. Un bon exercice : prenez quelques couples d'entiers et comparez les deux procédés.

Analyse

	NOM : calcul_pgcg	
S	**L.D.E.**	**O.U.**
3	**Résultat = Écrire** ("PGCD(",m,", ",n,") =", pgcd)	pgcd
2	pgcd = [pgcd ←m] **Si** (m=0) **alors**	m
	pgcd ← n	n
	FinSi	
1	(mf,nf) = [m=Donnée("m = "),n=Donnée("n = ")]	
	Tant Que (m * n≠0) **Faire**	
	[ ] **Si** (m>n) **alors**	
	m ← m MOD n	
	sinon	
	n ← n MOD m	
	FinSi	
	FinTantQue	
4	**Fin calcul_pgcd**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
pgcd	Entier	Le PGCD de m et n
m	Entier	Le premier entier
n	Entier	Le second entier

Algorithme

0) Début calcul_pgcd
1) [Lire("m = ",m),Lire("n = ", n)]
*Tant Que (m*n ≠0) Faire*
[ ] Si (m>n) alors
m ← m MOD n
sinon
n ← n MOD m
FinSi
FinTantQue
2) [pgcd ←m] Si (m=0) alors
pgcd ← n
FinSi
3) Écrire ("PGCD(",m,", ",n,") =", pgcd)
4) Fin calcul_pgcd

Traduction en Pascal

```
PROGRAM calcul_pgcd;
USES WINCRT;
VAR  m, n, pgcd : INTEGER;

BEGIN
  WRITE('m = '); READLN(m);
  WRITE('n = '); READLN(n);
  WHILE (m*n<>0) DO
  BEGIN
    IF (m>n) THEN
       BEGIN
          m:= m MOD n;
       END
    ELSE
       BEGIN
          n:= n MOD m;
       END;
    END;
  pgcd:=m;
  IF (m=0) THEN
    BEGIN
       pgcd:=n;
    END;
  WRITELN('PGCD(',m,',',n,') = ',pgcd);
END.
```

Un cas d'exécution

```
m = 25
n = 15
PGCD(0,5) = 5
```

Remarque :

Dans ce problème, nous avons adopté la formulation tant que car il est possible de ne pas traiter l'objet d'itération. Référerez-vous à la deuxième activité introductrice où le convoyeur est contraint de ne pas faire monter des voyageurs s'il n'y a plus de places assises.
En effet dans cette deuxième formulation (tant que), la condition est vérifiée avant d'attaquer le traitement à répéter, contrairement à la première formulation (répeter.. jusqu'à) où la condition est vérifiée après avoir exécuté une fois le traitement.
Attaquons maintenant une activité mathématique d'approximation.

Activité 7

En mathématiques, en fait, on sait résoudre peu d'équations. Par contre, il existe des théorèmes forts permettant d'assurer l'existence et parfois l'unicité d'une solution à une équation donnée. Nous en citons le théorème des valeurs intermédiaires. Exemple, dans l'intervalle [0,1], l'équation $e^{-x} - x = 0$, a une solution unique. Elle est mise en évidence en traçant les deux courbes d'équations respectives $y=e^{-x}$ et $y=x$. On se propose d'écrire un programme permettant de trouver et d'afficher une valeur approchée de cette solution à une erreur e près.

Analyse

On définit la fonction f telle que $f(x)=e^{-x} - x$ sur l'intervalle [a,b]=[0,1] (a=0 et b=1). Nous allons utiliser la méthode de dichotomie pour déterminer cette solution. Elle consiste à calculer l'image du milieu de l'intervalle en cours [a,b], on compare son signe avec f(a) en utilisant leur produit. Si ce dernier est nul, c'est en fait, l'image du milieu qui l'est et dans ce cas le problème est fini, nous avons trouvé la solution. S'il est négatif, la solution est entre a et c, il suffit de remplacer b par c? S'il est positif, la solution est dans l'autre moitié, il faut donc remplacer a par c. On continue ce travail tant que la différence en valeur absolue de a et b est supérieur à 2*e. La solution cherchée sera égale au milieu de a et b.

	NOM : calcul_racine	
S	**L.D.E.**	**O.U.**
2	**Résultat = Écrire** ("La racine est : ", (a+b)/2, " à ", e, " près.")	a
1	(a,b) = [a←0, b←1, Lire("e = ",e)]	b
	TantQue (|a-b| >2*e) **Faire**	e
	[c← (a+b)/2, p← (exp(-a)-a)(exp(-c)-c)] **Si** (p=0) **alors**	c
	a←c	p
	b←c	
	Sinon Si (p<0) **alors**	
	b←c	
	Sinon	
	a←c	
	FinSi	
	FinTantQue	
3	**Fin calcul_racine**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
a	Réel	La borne inf de l'intervalle de recherche
b	Réel	La borne sup de l'intervalle de recherche
e	Réel	La valeur de l'erreur
c	Réel	La valeur du milieu de [a,b]
p	Réel	Comporte le produit de f(a).f(c)

Algorithme

0) Début calcul_racine
1) [a← 0, b← 1,Lire("e = ",e]
TantQue (|a-b| >2.e) Faire
[c← (a+b)/2, p← (exp(-a)-a)(exp(-c)-c)] Si (p=0) alors
a←c
b←c
Sinon Si (p<0) alors
b←c
Sinon
a←c
FinSi
FinTantQue
3) Écrire ("La racine est : ", (a+b)/2, " à ", e, " près.")
4) Fin calcul_racine

Traduction en Pascal

```
PROGRAM calcul_racine;
USES WINCRT;
VAR     a, b, e, c,p : REAL;

BEGIN
  a:=0;
  b:=1;
  WRITE('e = '); READLN(e);
  WHILE (ABS(a-b)>2*e) DO
    BEGIN
      c:=(a+b)/2;
      p:=(EXP(-a)-a)*(EXP(-c)-c);
      IF p=0 THEN
                BEGIN
                  a:= c;
                  b:=c;
                END
              ELSE IF (p<0) THEN
                                BEGIN
                                     b:=c;
                                END
                              ELSE
                                BEGIN
                                     a:=c;
                                END;
    END;
  WRITELN('La racine est :',(a+b)/2:6:3,'à',e:6:3,' près.');
END.
```

Un cas d'exécution

```
e = 0.001
La racine est : 0.567 à  0.001 près.
```

Retenons

-Il existe deux formulations pour traduire une structure itérative à condition d'arrêt:

- La structure Répéter … Jusqu'à …
- La structure Tant Que … Faire …

- Dans la structure "Répéter …Jusqu'à …" , le traitement à répéter est exécuté au moins une fois car la condition d'arrêt est testé à posteriori.
- Dans la structure "Tant Que … Faire …. " le traitement à répéter peut ne pas être exécuté car la condition d'arrêt est testée à priori. En effet, dans le cas où elle est vraie dès le début, le traitement n'est pas exécuté.
- Dans l'une ou l'autre formulation, il est conseillé de vérifier à la main l'exécution pour s'assurer du bon fonctionnement de la boucle et surtout des cas limites. Il est important de bien initialiser la structure et de bien positionner les instructions à l'intérieur de la boucle.

Exercices

Exercice 1

Donner la différence entre les deux formulations traduisant une structure itérative à condition d'arrêt.

Exercice 2

Quand il s'agit de faire un contrôle lors de la saisie d'une donnée, pourquoi il est préférable d'opter pour la structure "Répéter .. Jusqu'à…"

Exercice 3

Analyser puis déduire un algorithme et le programme Pascal qui permet de chercher si une valeur v existe dans un tableau. Dans l'affirmative, le programme affiche la dernière et l'avant dernière place si elle existe.

Exercice 4

Écrire un programme qui saisit un mot en français puis il affiche la première lettre double dans l'écriture de ce mot.

Exemple :

ch = "programme"

Le programme affiche la lettre m.

Exercice 5

Écrire un programme qui saisie une phrase en français puis affiche le nombre de mots de cette phrase.

Exercice 6

Écrire un programme qui calcule le PGCD de deux entiers en utilisant la méthode de la différence.

Exercice 7

Le PPCM de deux entiers est le plus petit commun multiple des deux entiers m et n. Pour calculer le PPCM de m et n, on pourra utiliser la formule suivante :

PPCM(m,n) = m.n/PGCD(m,n)

Exercice 8

Deux entiers naturels non nuls m et n sont premiers entre eux si et seulement si ils ont un PGCD égal à 1.

Exemple :

Pour **m = 7** et **n = 4**, PGCD (7,4) = 1 donc **7** et **4** sont premiers entre eux.

En utilisant cette propriété, écrire un programme permettant de saisir deux entiers naturels non nuls m et n et d'afficher s'ils sont premiers entre eux ou non.

Exercice 9

Écrire un programme qui saisit un mot et d'en extraire la première partie du mot comportant plus de voyelles que de consonnes et de l'afficher.

Exercice 10

On dispose de deux tableaux T1 et T2 contenant respectivement n et m entiers positifs et non nuls.

On désire chercher si les éléments de T1 sont tous dans T2 ou inversement. On dira que "T1 est inclus dans T2" ou "T2 est inclus dans T1" ou "T1 et T2 non comparables"

Analyser puis déduire un algorithme et le programme Pascal qui saisit m et n et remplit deux tableaux T1 et T2 respectivement par m et n réels. Ensuite étudie la relation entre T1et T2 comme indiqué plus haut et affiche le résultat.

Exercice 11

Soit l'algorithme suivant :

```
0) Début inconnu
1) Écrire (" n = "); Lire (n)
2)   Pour i de 1 à n Répéter
        Répéter
           Lire(T[i])
        Jusqu'à T[i] Dans [0..9]
   FinPour
3) Écrire ("La plus longue croissante de T mesure ",plc)
4) Fin Inconnu
```

Questions :

1. Traduire cet algorithme en Pascal.
2. Dans **l'action 1**, ajouter les contrôles sur la saisie pour que **n** vérifie la condition suivante : 5 ≤ n ≤ 50
3. Écrire une séquence d'instructions permettant de déterminer plc, la longueur de la plus longue séquence croissante d'éléments du tableau.

Exemple : Pour le tableau T suivant :

La séquence à chercher doit retourner la valeur 3 dans ce cas.

T	2	4	3	**1**	**5**	**7**	2	8	**4**	**8**	**9**	**0**
	1	2	3	4	5	6	7	8	9	10	12	13

Exercice 12

Nous savons par son graphique qu'une fonction donnée f admet un maximum en x_0 dans un intervalle [a,b]. On se propose de calculer une valeur approchée de x_0 à e près où e est une erreur fixée d'avance. Pour ce faire, on calcule l'image f(x) pour une suite de x à un pas égal à 1/2 e jusqu'à ce que f change de monotonie. Les deux dernières valeurs de x encadrent convenablement le x_0 recherché.

Analyser puis déduire un algorithme et le programme Pascal qui détermine à e près (0.01 par exemple) une valeur approchée de x_0 où la fonction suivante f admet un minimum.

f(x) = x+1 +1/x sur l'intervalle]0,4]

CHAPITRE

Chapitre 5

Les sous-programmes

Objectifs :

- Décomposer un problème en sous-problèmes élémentaires

- Présenter les solutions sous forme de procédures et de fonctions

- Présenter les solutions sous forme d'algorithmes puis de programmes.

Plan du chapitre :

Leçon 1

L'analyse modulaire

Objectifs spécifiques :

- Comprendre l'intérêt de l'analyse modulaire
- Utiliser l'analyse modulaire pour résoudre des problèmes

Plan de la leçon :

I. Introduction

II. L'analyse modulaire

- **II.1.** Définition
- **II.2.** Intérêts
- **II.3.** Notion de sous-programme
- **II.4.** Exemple d'analyse modulaire

Retenons

Exercices

Leçon 1

L'analyse modulaire

Diviser chacune des difficultés que j'examinerais en autant de parcelles qu'il se pourrait et qu'il serait requis pour les mieux résoudre
DESCARTES

I. Introduction

Dans le cas de la résolution d'un problème simple, il est possible d'écrire directement l'algorithme correspondant. Toutefois, dans le cas d'un problème plus complexe, la solution nécessiterait l'écriture de plusieurs sous-programmes résolvant chacun une partie du problème.

En général, et afin de faciliter la résolution d'un problème complexe et/ou de grande taille, on doit le décomposer en sous-problèmes indépendants et de taille moindre. Ce principe de réduction de complexité est connu sous le nom : « Diviser pour régner ».

Exemple 1 :
Soit le problème consistant en la détermination de tous les nombres premiers compris entre 1 et 10000. La résolution de ce problème nécessiterait de développer un sous programme permettant de décider si un nombre entier donné est premier. La solution globale fera alors appel à ce sous-programme (module) pour tous les nombres entiers de la fourchette 1 à 10000.

Exemple 2 :
Votre classe veut organiser une fête de fin d'année. Il est alors nécessaire de se partager le travail afin de réussir ce projet. Dans ce but, un chef de projet et des groupes d'élèves seront constitués. Chaque groupe aura une tâche précise et devra la programmer. Un groupe s'occupe de la décoration de la salle ; un autre s'occupe des invités ; un autre s'occupe des prix à distribuer aux bons élèves et un autre groupe prend en charge l'aspect musical. Le chef de projet pilote l'ensemble des groupes. Il appelle chaque groupe pour l'exécution d'une tâche donnée au moment opportun.

À travers cet exemple, des notions importantes apparaissent suite à des questions naturellement posées. En effet, le chef de projet aura à communiquer avec chaque groupe de travail en précisant des données comme par exemple les dimensions de la salle et il obtiendra des résultats comme par exemple le nombre des personnes qu'elle peut contenir. Le calcul du nombre de personnes est bien sûr élaboré par un des groupes de travail en utilisant des ressources locales telles que les mesures des tables, etc.… Outre ces communications, les groupes de travail peuvent utiliser quelques ressources du chef du projet ; ça peut être par exemple une caisse commune.

Toutes ces notions sont explicitées par la suite dans ce chapitre où nous présentons encore une fois une introduction à l'analyse modulaire, la notion de module ou de sous-programme et les problèmes qu'elle pose quant à la transmission des paramètres et le partage des données. Nous explicitons les cas des fonctions et des procédures.

II. L'analyse modulaire

Un aspect important d'une méthodologie d'analyse est de rendre possible la décomposition modulaire d'un programme en plusieurs sous-programmes. Il est certain que toute méthode d'analyse qu'elle soit descendante ou ascendante, voit son efficacité renforcée si on la combine avec l'utilisation de cette notion.

II.1 Définition

> **L'analyse modulaire consiste à diviser un problème en sous problèmes de difficultés moindres. Ces derniers peuvent être aussi assujettis à cette division jusqu'à ce qu'on arrive à un niveau abordable de difficulté.**

II.2 Intérêts

Le recours à une méthode d'analyse utilisant le concept de la modularité présente un ensemble d'avantages. Parmi ceux-ci, on cite :

- Améliorer l'écriture de l'algorithme de résolution et du programme associé en évitant les duplications de textes. Si une même suite d'instructions figure à plusieurs endroits d'un programme, elle pourra constituer un sous-programme et être remplacée par un appel à ce sous-programme.
- Pouvoir poursuivre l'analyse du problème initial comme si différents sous-problèmes étaient résolus ; chaque sous-problème lui est alors associé un sous-programme.
- Faciliter la résolution du problème en ne s'intéressant qu'à la résolution algorithmique d'un sous-problème à la fois.
- Décrire l'algorithme de chaque sous-programme avec un tableau de déclaration indépendant mais relatif à l'environnement du sous-problème résolu.
- Réutiliser des sous-programmes par ailleurs décrits tels que des fonctions appartenant à des bibliothèques standard (telle que la bibliothèque de fonctions mathématiques, la bibliothèque de fonctions de gestion de l'écran, etc.…).

Ainsi, le programme sera plus lisible, plus facile à documenter, à mettre au point et à maintenir (modifier éventuellement par la suite).

Si la décomposition modulaire est un bon moyen de structurer les programmes, il faut que le découpage soit significatif : un sous-programme doit correspondre à une tâche déterminée qui intervient "naturellement" dans l'analyse du problème. L'analyse modulaire met en évidence progressivement des sous-problèmes : ceci offre "naturellement" la possibilité de définir des sous-programmes de tâches claires, précises et explicites.

II.3. Notion de sous-programme

C'est une portion de code analogue à un programme, déclaré dans un programme ou dans un sous-programme et dont la partie instruction peut être exécutée plusieurs fois au cours du traitement du programme grâce à des appels.

On distinguera alors les sous-programmes appelant et appelé. L'appelant va activer l'appelé à l'aide d'une instruction d'appel et lorsque l'exécution de l'appelé cesse, un retour à l'appelant s'effectue afin de poursuivre l'exécution à partir de l'action qui suit l'appel.

On distingue deux types de sous-programmes : les fonctions et les procédures. Une fonction réalise un traitement et retourne un seul résultat, comme par exemple la fonction factoriel(n) alors qu'une procédure réalise un traitement et pourrait retourner zéro, un ou plusieurs résultats. A titre d'exemple, nous citons la procédure de tri d'un tableau d'entiers (voir chapitre 5).
Décider si un sous-programme est une fonction ou une procédure se fait lors de l'analyse et ce, en fonction de la nature du traitement et des entrées/sorties réalisées.
Ces deux types de sous-programmes feront l'objet des leçons suivantes.

II.4. Exemple d'analyse modulaire

Nous proposons dans l'exemple ci-dessous non pas une solution complète du problème mais plutôt une approche de résolution du problème.

PROBLÈME

Donner un algorithme pour la construction d'une maison.
Arrêtez-vous à ces énoncés et ne regardez pas la solution proposée. Prenez une feuille et un crayon, essayez de tracer une démarche de résolution de ce problème. Puis essayez de comparer votre proposition avec celles de vos camarades.

Solution :
Afin d'aborder ce problème, il est recommandé d'appliquer le principe de réduction de complexité (diviser pour régner). Ceci nous conduit à adopter une approche modulaire et descendante afin de diviser le problème en sous problèmes ou en modules de difficulté moindre. Si dans un module, il y a encore davantage de complexité, on continue à le diviser en sous modules et ainsi de suite jusqu'à arriver à un niveau élémentaire de difficulté.

Nous devons d'abord reposer le problème en détails. Nous supposons que le futur propriétaire de cette maison va faire le choix du terrain et de son emplacement. Il a aussi une idée claire et précise du plan de la maison. Commençons donc l'analyse de ce sujet.

Analyse

	NOM : ConstructionMaison	
S	**L.D.E.**	**O.U.**
1	**Résultat = MaMaison** MaMaison = (MODULE Finition, MODULE Toits MODULE Murailles MODULE Fondations MODULE Terrain)	Finition Toits Murailles Fondations Terrain
2	**Fin ConstructionMaison**	

Nous constatons que le résultat MaMaison est une suite ordonnée de cinq modules qui sont : Finition, Toits, Murailles, Fondations et Terrain. La disposition de ces modules a été carrément descendante.

Nous passerons ensuite à l'analyse de chaque module à part. Prenons l'exemple du module Finition.

	MODULE Finition	
S	**L.D.E.**	**O.U.**
1	**Résultat = MaisonFinie** MaisonFinie = (MODULE DernierNettoyage, MODULE Peinture, MODULE Badigeonnage MODULE Sanitaires MODULE Electricité	DernierNettoyage Peinture Badigeonnage Sanitaires Electricité
2	Etc ... **Fin** MODULEFinition	Etc.

Vous avez remarqué à travers cet exemple comment diviser un grand programme de construction d'une maison en sous-programmes. Eux-mêmes, ces sous-programmes sont divisés en sous-programmes. C'est l'approche modulaire.

Retenons

Pour faciliter la résolution des problèmes, souvent, ils sont décomposés en sous-problèmes par une analyse modulaire. Chaque sous-problème fait l'objet d'une procédure ou d'une fonction.

Exercices

Exercice 1

On veut saisir le contenu de deux vecteurs d'entiers A et B et les réunir dans un vecteur C que l'on affichera par la suite. Décomposer ce problème en en un ensemble de sous-problèmes et analyser chacun de ces sous-problèmes.

Exercice 2

Nous avons relevé la pluviométrie mensuelle concernant les 5 dernières années. Nous voulons déterminer la moyenne pluviométrique annuelle, les pluviométries mensuelles minimale et maximale. Décomposer ce problème en un ensemble de sous-problèmes et analyser chacun de ces sous-problèmes.

Exercice 3

On veut écrire un programme qui permet de :

- initialiser automatiquement un tableau de dimension N avec des valeurs aléatoires sachant que pour générer des nombres aléatoires on doit utiliser :
  - la procédure RANDOMIZE ; sans paramètres, elle initialise le générateur des nombres aléatoires et doit être mise au début du programme. Il est à noter que la procédure RANDOMIZE appartient à la bibliothèque WINCRT .
  - la fonction prédéfinie RANDOM (P) ; qui envoie un entier aléatoire Z tel que $0 \leq Z \leq P$ où P est un entier.

- tester si le tableau constitue un palindrome (on obtient le même résultat quand on lit un palindrome de gauche à droite ou de droite à gauche).

Décomposer ce problème en un ensemble de sous-problèmes et analyser chacun de ces sous-problèmes.

Exercice 4

Soit un tableau T d'entiers non nuls, trouvez combien il y a d'entiers distincts positifs dont l'opposé est aussi dans le tableau.

Exemple : Donnée T : -3 4 2 8 9 1 -3 -8 -4 2 8 2 -8 1 3
Résultat : 3

On affichera 3 car les trois entiers 3, 4, et 8 ont leurs opposés dans le tableau T.

Questions :

1. Analyser ce problème en le décomposant en modules
2. Analyser chacun de ces modules

Exercice 5

Le jeu se joue à deux joueurs :

• Le premier joueur saisit un entier de quatre chiffres puis l'affiche masqué avec des étoiles

• Le deuxième joueur se charge de dévoiler cet entier :

- il saisit un entier, au hasard, de quatre chiffres
- l'ordinateur va l'aider en indiquant si cette valeur est plus grande, plus petite ou égale à l'entier masqué
- Ce joueur répète ces deux étapes jusqu'à dévoiler l'entier masqué par le premier joueur.

• Lorsque le deuxième joueur devine quel entier il s'agit, le programme affiche le nombre d'essais qui ont été nécessaires à ce joueur pour dévoiler cet entier.

Questions :

1. Analyser ce problème en le décomposant en modules
2. Analyser chacun de ces modules

Exercice 6

Dans un tableau de N éléments, le programme multiplie les nombres des cases impaires par 2 et des cases paires par le contenu de la case précédente. Le programme s'arrête en fin de tableau ou lorsqu'il rencontre une case de valeur 0.

7	6	5	9	4	0	0	0	0	**AVANT**
1	2	3	4	5	6	7	8	9	
14	48	10	18	36	0	0	0	0	**APRÈS**

Questions :

1. Analyser ce problème en le décomposant en modules
2. Analyser chacun de ces modules

Exercice 7

Soit N un entier naturel non nul, on se propose de déterminer le plus grand entier formé par tous les chiffres de N.

Exemple : N= 3941 → l'entier résultat est 9431
N= 65060 → l'entier résultat est 66500

Questions :

1. Analyser ce problème en le décomposant en modules.
2. Analyser chaque module.

Leçon 2

Les fonctions

Objectifs spécifiques

- Comprendre l'intérêt des sous-programmes de type fonction
- Savoir déclarer, programmer et utiliser les fonctions

Plan de la leçon

Leçon 2

Les fonctions

Ce sont ces questions, qui laissent une partie de la proposition en blanc, que les mathématiciens appellent problèmes, comme lorsqu'on demande de trouver un miroir qui ramasse tous les rayons du soleil en un point, c'est à dire, on en demande la figure ou comment il est fait.

LEIBNIZ

I. Définition

Une fonction est un sous-programme qui renvoie une valeur d'un seul type. Ce type sera celui de la fonction.

La communication entre l'appelant et la fonction (ici l'appelé) se produit à l'aide d'un ensemble de paramètres et éventuellement de ressources communes partagées. Ceci sera détaillé dans la leçon 3.

II. Vocabulaire et syntaxe

Ce paragraphe présente de manière progressive et illustrée le vocabulaire et la syntaxe utilisés lors de l'utilisation des fonctions tant au niveau de l'appel qu'au niveau de la déclaration de la fonction.

Activité 1

Soit la fonction f(x)=4.x+0,75. Nous voulons écrire un programme qui affiche pour n valeurs de x, le couple (x, f(x)), x étant une valeur donnée.

Analyse

	Nom : Calcul_F	
S	**L.D.E.**	**O.U.**
1	**Résultat** = valeurs Valeurs = [n = Donnée] **Pour** i de 1 à n **faire** x = Donnée ("Entrer la valeur de x ") y ← FN f(x) Écrire (x, y) **Fin**Pour	n i x y f
2	**Fin Calcul_F**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
n	Entier	Nombre d'itérations
i	Entier	Compteur
x	Réel	Abscisse
y	Réel	Valeur de f(x)
f	Fonction de valeur f(x) =4*x+0.75	Fonction

Algorithme

0) Début Calcul_F
1) [Lire (n)] ***Pour*** *i de 1 à n* ***Faire***
Ecrire ("Entrer la valeur de x ")
Lire (x)
y ← FN f(x)
Ecrire (x, y)
FinPour
2) Fin Calcul_F

Remarques :

1) L'appel de la fonction f est fait à l'aide de FN f(x).

2) Le préfixe FN est ajouté devant le nom de la fonction que nous avons créée ; cette convention nous aidera à repérer les fonctions à définir.

Analyse de la fonction f

	DEF FN f (x : Réel) : Réel	
S	**L.D.E.**	**O.U.**
1 2	**Résultat = f** f ← 4*x + 0.75 **Fin f**	

Remarque :

x et f sont déjà déclarés au niveau de l'entête de la fonction (DEF FN f (x : Réel) : Réel).

Algorithme

0) DEF FN f (x : Réel) : Réel
*1) f ← 4*x+0.75*
2) Fin f

Remarque :

f est une fonction ayant un seul paramètre x. Il est possible qu'une fonction ait plusieurs paramètres. Dans l'activité ci-dessous, la fonction Moyenne en possède plusieurs.

Activité 2

Écrire un programme qui saisit deux coefficients coef1 et coef2 puis répète n fois la saisie de deux notes Note1 et Note2 et appelle une fonction pour calculer leur moyenne pondérée. Cette moyenne sera ensuite affichée.
On utilise la formule : moyenne = (Coef1.Note1 + Coef2.Note2)/ (Coef1+Coef2).

Analyse

	Nom : Calcul_moy	
S	**L.D.E.**	**O.U.**
3	**Résultat** = Les_moyennes Les_moyennes = [Lire("n = ",n)]**Pour** i de 1 à n **faire** Note1 = Donnée("Note1 = ") Note2 = Donnée("Note2 = ") Moy ← FN Moyenne (Coef1, Note1, Coef2, Note2) Ecrire ("Moyenne = ", Moy) **FinPour**	n i Note1 Note2 Moy Moyenne Coef1 Coef2
1	Coef1 = Donnée ("Donner le 1er coefficient")	
2	Coef2 = Donnée ("Donner le 2ème coefficient")	
4	**Fin Calcul_moy**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
n	Entier	Nombre de calculs de moyennes
i	Entier	Compteur
Note1	Réel	1ère note
Note2	Réel	2ème note
Moy	Réel	Moyenne pondérée des 2 notes
Moyenne	Fonction	Fonction qui calcule la moyenne pondérée des 2 notes
Coef1	Entier	Coefficient de la 1ère matière
Coef2	Entier	Coefficient de la 2ème matière

Algorithme

0) Début Calcul_moy
1) Écrire ("Donner le 1er coefficient : "), Lire (Coef1)
2) Écrire ("Donner le 2ème coefficient : "), Lire (Coef2)
3) [Lire ("n = ",n)] ***Pour*** *i de 1 à n* ***Faire***
Lire ("Note1 = ", Note1)
Lire ("Note2 = ", Note2)
Moy ← FN Moyenne (Coef1, Note1, Coef2, Note2)
Écrire ("Moyenne = ", Moy)
FinPour
4) Fin Calcul_moy

Remarque :

L'appel de la fonction qui calcule la moyenne des deux notes est réalisé à l'aide de FN Moyenne (Coef1, Note1, Coef2, Note2).

Analyse de la fonction Moyenne

	DEF FN Moyenne (c1, n1, c2, n2 : Réel) : Réel	
S	**L.D.E.**	**O.U.**
2 1 3	**Résultat** = Moyenne Moyenne ← Total / (c1 + c2) Total ← c1*n1 + c2*n2 **Fin Moyenne**	Total

Remarque :

Nous constatons que la liste des objets utilisés (O.U.) n'inclut que Total. En effet, les autres objets (c1, c2, n1, n2) sont déclarés dans l'entête de la fonction en tant que paramètres. L'objet Moyenne constitue en soi, l'identificateur de la fonction, déclaré aussi dans l'entête de la fonction. Il constitue le résultat de retour de la fonction et il est du type réel.

Tableaux de déclaration des objets locaux

Objet	Type/nature	Rôle
Total	Réel	Total pondéré des moyennes

Algorithme

0) DEF FN Moyenne (c1, n1, c2, n2 : Réel) : Réel
*1) Total ← c1*n1 + c2*.n2*
2) Moyenne ← Total / (c1 + c2)
3) Fin Moyenne

Remarque :

Nous remarquons aussi que dans le tableau de déclaration des objets, seule la variable locale Total est déclarée.
Les paramètres c1, n1, c2 et n2 appelés paramètres formels (voir leçon 3) sont déjà déclarés dans l'entête de la fonction.

Traduction en Pascal

```
PROGRAM Calcul_moy ;
USES WINCRT ;
VAR     Coef1, Coef2, i, n :INTEGER ;
        Note1, Note2, Moy : REAL ;
FUNCTION Moyenne ( c1,n1,c2,n2 : REAL) : REAL ;
VAR Total : REAL ;
BEGIN
  Total := c1 * n1 + c2 * n2 ;
  Moyenne := Total / ( c1+c2 );
END ;
BEGIN  { PROGRAMME PINCIPAL}
        WRITELN('Entrer le 1 er coefficient : ') ;
        READLN (Coef1) ;
        WRITELN('Entrer le 2ème coefficient : ') ;
        READLN (Coef2) ;
        WRITE('n = ');
        READLN (n) ;
        FOR i :=1 TO n DO
            BEGIN
              WRITE('Note1 = ');
              READLN (note1) ;
              WRITE('Note2 = ');
              READLN (note2) ;
              Moy :=Moyenne(Coef1, Note1,Coef2, Note2) ;
              WRITELN ('Moyenne = ',Moy:5:2);
            END ;
END.
```

Un cas d'exécution

```
Entrer le 1 er coefficient :
3
Entrer le 2ème coefficient :
2
n = 2
Note1 = 11.5
Note2 = 10
Moyenne = 10.90
Note1 = 13
Note2 = 16
Moyenne = 14.20
```

Remarque :

La déclaration d'une fonction en PASCAL se trouve dans la partie déclarative d'un programme ou d'un sous-programme, après les déclarations des constantes, des types et des variables.

Retenons

Une fonction est constituée de trois parties :

1) La partie "entête de la fonction" où nous trouvons le nom de la fonction suivi entre parenthèses des paramètres en entrée et de leur mode de passage, puis du type du résultat.

2) La partie "déclaration locale" où tous les objets locaux de la fonction sont déclarés.

3) La partie "corps de la fonction" où nous trouvons les instructions propres à la fonction.

Exercices

Exercice 1

a- Écrire une fonction MIN2 qui donne le minimum de deux entiers.
b- Écrire une fonction MIN3 qui donne le minimum de trois entiers.

Exercice 2

Écrire un programme intitulé AMI qui permet d'afficher si deux entiers m et n sont deux nombres amis. m et n sont dits amis si la somme des diviseurs de m est égale à la somme des diviseurs de n.

Exercice 3

On veut écrire un programme permettant de lire deux mots ch1 et ch2, d'appeler une fonction qui prend en paramètres ch1 et ch2 et retourne une chaîne de caractères composée par tous les caractères qui apparaissent dans les deux chaînes sans redondance et d'affiché le résultat de la fonction.

Exemple :

Soit ch1= "Coccinelle" et ch2= "Cible" Résultat : "C i l e"

Exercice 4

La factorielle d'un entier N est définie par N ! =1*2*3*…*N et 0 ! = 1
Écrire un programme qui permet de calculer la somme suivante :
S = 1-1/3 !+1 /5 !-1/7 !+ ……1/N !
Reconnaissez-vous le nombre trouvé ?

Exercice 5

Un entier naturel est dit parfait s'il est égal à la somme de tous ses diviseurs autres que lui même.

Exemple : 6 = 1+2+3

On demande d'écrire un programme qui cherche et affiche tous les entiers parfaits compris entre 2 valeurs données M et N (2<M<N).

Exercice 6

Écrire une fonction permettant de calculer le PGCD(Plus Grand Commun Diviseur) d'un couple d'entiers naturels (a,b) en utilisant les deux formules suivantes :
1)PGCD(a,b)=PGCD(b ,a MOD b) ;
2)PGCD(a,b)=PGCD(b,a-b) ;

Exercice 7

Écrire une fonction qui calcule la racine carrée d'un réel positif a en utilisant la limite de la suite$(x_n)_{n \in N}$ définie par : $x_{n+1} = 1/2 * (x_n + a/x_n)$
x_0 étant choisi arbitrairement

Exercice 8

$P(x) = a_0 + a_1 * x^2 + a^2 * x^3 + \ldots + a_n x^n$ étant un polynôme de degré n. Le schéma de Horner P(x) est : $P(x) = a_0 + x*(a_1 + x*(a_2 + x*(\ldots. a_n*x))) \ldots.)))$.
Écrire une fonction qui calcule la valeur de P(x) pour un x donné en utilisant le schéma de Horner.

Leçon 3

Declaration, accès aux objets et modes de transmission

Objectifs spécifiques

- Comprendre l'utilisation des objets locaux et des objets globaux.
- Comprendre et savoir utiliser les paramètres et leurs modes de transmission.

Plan de la leçon

Leçon 3

Declaration, accès aux objets et modes de transmission

Le bon sens est le concierge de l'esprit : son office est de ne laisser entrer ni sortir les idées suspectes.

Daniel STERN

I. Déclaration et accès aux objets

I.1 Les objets locaux

Dans la fonction Moyenne de la leçon précédente, la variable Total de type Réel est déclarée dans le tableau des déclarations des objets locaux de cette fonction. C'est une variable locale à la fonction.

Définition :

> **Tous les objets tels que les constantes, les types, les variables et éventuellement d'autres fonctions déclarés dans un sous-programme sont dits locaux à celui-ci.**

I.2 Les objets globaux

Définition :

> **Les objets utilisés dans un sous-programme et non déclarés dans celui-ci sont des objets globaux déclarés dans la zone déclaration du programme appelant.**

Activité 1

Soit le programme Pascal suivant, identifier les objets locaux et les objets globaux.

```
PROGRAM cercle ;
 CONST pi = 3.1415 ;
 VAR   rayon : REAL ;
       p, s : REAL;

 FUNCTION perimetre ( r : REAL) : REAL ;
```

```
BEGIN
   perimetre = 2 * pi * r ;
  END ;
        FUNCTION surface ( r : REAL) : REAL ;
  BEGIN
   surface = pi * r * r ;
  END ;

  BEGIN  { PROGRAMME PINCIPAL}
   WRITELN('Entrer le rayon du cercle : ') ;
   READLN (rayon) ;
   p := perimetre (rayon) ;
   s := surface (rayon) ;
   WRITELN('Le périmètre du cercle est : ', p) ;
   WRITELN('La surface du cercle est : ', s) ;
   END.

END.
```

Réponse :

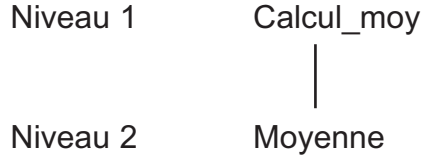
Les deux fonctions périmètre et surface utilisent l'objet global pi. Les objets rayon, p, et s déclarés dans le programme principal sont considérés comme des objets locaux à celui-ci.

I.3. Niveaux des sous-programmes

La déclaration des sous-programmes respecte une hiérarchie dictée par les différents niveaux d'appels. Cette dernière peut être représentée par un arbre où le premier niveau correspond à sa racine, les autres niveaux correspondent quant à eux aux diverses imbrications des sous-programmes entre eux.
Pour représenter un arbre, on considère que :

- sa racine est l'identificateur du programme principal.
- ses nœuds correspondent aux divers sous-programmes qui les composent.
- les arcs représentent la relation " déclare " ou " est déclaré par ".

L'arbre de l'algorithme Calcul_moy est simple et ne contient qu'un seul arc. Cet arbre est schématisé comme suit :



Remarque :

Tous les objets prédéfinis et notamment les fonctions telles que ABS et RacineCarré sont considérés au niveau 0. C'est le niveau visible en tout point du programme.

I.4. Accès aux objets

L'objectif de ce paragraphe est de présenter les règles d'accès aux objets dans une hiérarchie de sous programmes.

Tous les objets locaux d'un sous-programme sont **inaccessibles** :
- par le programme principal.
- par les sous-programmes déclarés au même niveau que le sous-programme considéré.
- par les sous-programmes qui englobent le sous-programme considéré.

Dans la fonction Moyenne de la leçon précédente, la variable locale Total de la fonction Moyenne n'est pas connue dans l'algorithme principal Calcul_moy.

La règle d'accès aux objets locaux peut se définir comme suit.

Définition :

> **Un objet déclaré dans un sous-programme S à un niveau i est inaccessible par les sous-programmes de niveau j < i et par les sous-programmes de niveau k > i qui ne sont pas englobés par S.**

Cette règle définit **la portée** de l'identificateur d'un objet. Elle peut être aussi formulée de cette façon :

Dans un sous-programme de niveau i, les identificateurs connus sont les identificateurs déclarés localement au sous-programme i et les identificateurs déclarés dans les sous-programmes englobant de niveau j < i.

Dans l'exemple précédent de la leçon II, les variables Coef1, Note1, Coef2 et Note2 sont déclarées dans le programme principal englobant la fonction Moyenne. Elles peuvent être utilisées sans nouvelles déclarations dans la fonction qui est un sous-programme englobé.

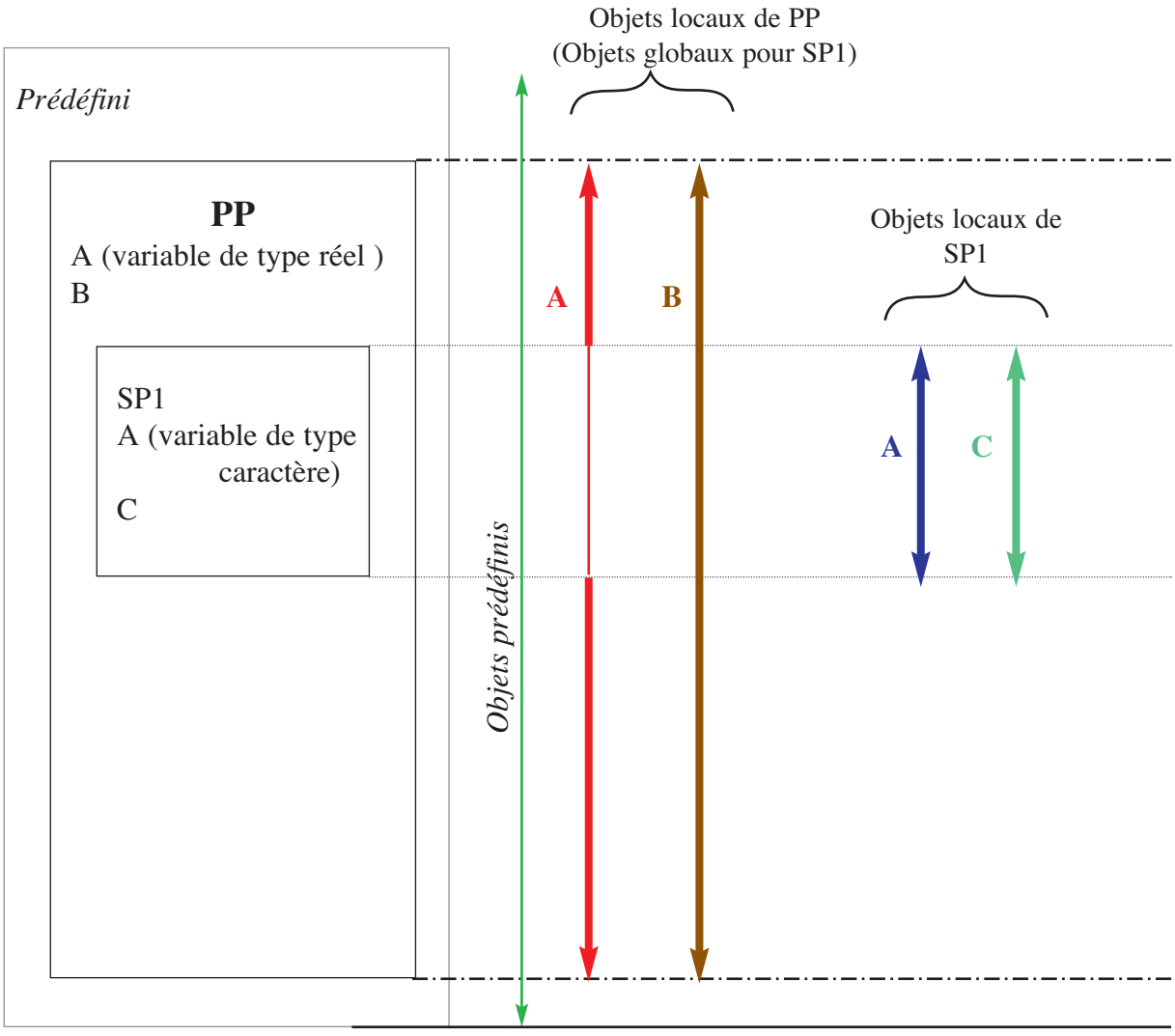
Ces objets sont alors appelés objets globaux.

Attention

« Un identificateur peut en cacher un autre ». C'est le cas où un identificateur d'un programme ou d'un sous-programme englobant a un homonyme (identificateur de même nom) dans un sous-programme englobé. Dans ce cas, l'objet local déclaré dans l'englobé va cacher l'objet global de même nom tout le long du sous-programme où il a été déclaré.
Nous venons de définir ainsi la règle de visibilité des objets :
La visibilité d'un objet (identificateur pouvant être utilisé) est égale à la portée moins la partie où il est caché par un homonyme.

Exemple récapitulatif :

Pour expliquer les différentes notions étudiées précédemment, nous allons utiliser l'exemple schématique suivant :



Commentaires :

1) La portée d'un objet est représentée par la flèche et sa visibilité est représentée par la partie en gras de cette même flèche.

2) Visibilité de la variable A (réel) = portée de A (réel) - la partie où A est cachée (portée de la variable locale A (caractère) du sous-programme SP1).

3) L'objet B est un objet global ; nous pouvons aussi l'utiliser dans le sous-programme SP1.

4) Dans notre schéma, le sous-programme SP1 est englobé ("fils") du programme principal PP qui est un englobant ("père") par rapport à SP1.
Dans cet état d'esprit, nous remarquons que "le fils" peut manipuler les objets définis " du père" (ou du "grand-père" …) mais "le père" n'a pas le droit de manipuler les objets définis par "le fils" (ou "petit fils"…).
PP ne peut pas manipuler les objets définis par SP1 ("fils") car il est de niveau inférieur.

Attention !

Afin d'écrire les sous-programmes les plus indépendants possibles, il faut éviter au maximum l'utilisation des variables globales non déclarées dans le sous-programme englobé (l'appelé). Néanmoins, on pourra éventuellement utiliser des objets globaux tels que des déclarations de types ou de constantes. Les sous-programmes devront de préférence communiquer à travers des paramètres.

II. Les paramètres et leurs modes de transmission

II.1. Les paramètres formels

Les paramètres formels sont les paramètres qui se trouvent dans l'entête du sous-programme.
Reprenons l'entête de la fonction Moyenne (leçon II) :

DEF FN Moyenne (c1, n1, c2, n2 : Réel) : Réel

Dans l'entête de la fonction Moyenne, c1, n1, c2 et n2 sont des paramètres formels. Pour leur spécification, il faut indiquer le nom, le type et le mode de passage de chaque paramètre (voir ci-dessous). Dans la partie instruction du sous-programme, on fait référence aux paramètres formels.
Dans la fonction Moyenne, c1, n1, c2 et n2 sont manipulés dans la partie instruction.

II.2. Les paramètres effectifs

Les paramètres effectifs sont les paramètres qui figurent (dans le programme ou sous-programme appelant) au niveau de l'appel du sous-programme. Leurs valeurs respectives sont relatives à une activation particulière et se substitueront aux paramètres formels. Le nombre des paramètres effectifs et des paramètres formels correspondants doit être le même. La correspondance entre paramètre formel et paramètre effectif s'établit d'après leur position dans la liste des paramètres. Il faut bien sûr s'assurer de la compatibilité de type entre paramètre effectif et paramètre formel correspondant. Dans l'activité2 (leçon II), Moyenne (Coef1, Note1, Coef2, Note2) constitue l'appel de la fonction Moyenne. Les objets Coef1, Note1, Coef2 et Note2 sont des paramètres effectifs.
Coef1 est un paramètre effectif correspondant au paramètre formel c1.
Note2 est un paramètre effectif correspondant au paramètre formel n2.
Coef2 est un paramètre effectif correspondant au paramètre formel c1.
Note2 est un paramètre effectif correspondant au paramètre formel n2.

II.3. Mode de passage des paramètres

Pour les sous-programmes, il existe deux modes de passage des paramètres : le mode de passage par valeur et celui par variable. Pour le cas de la fonction, nous définissons seulement le mode de passage par valeur lequel est le plus utilisé.
Le mode de passage par variable est expliqué dans la définition des procédures (voir paragraphe 3 de la leçon 4).

Mode de passage par valeur

Le paramètre effectif est une expression (on rappelle que ça peut être une constante, une valeur explicite, une variable ou un calcul simple donnant une valeur). Le paramètre formel correspondant désigne une variable locale du sous-programme. Au moment de l'appel, l'expression représentant le paramètre effectif est évaluée dans le contexte du programme ou du sous-programme appelant. La valeur du paramètre effectif est copiée dans la variable locale désignée par le paramètre formel correspondant.
L'appelant impose à l'appelé une pure lecture des valeurs des paramètres effectifs. Les paramètres transmis par valeur font communiquer l'information dans un seul sens: du sous-programme (ou programme)appelant vers le sous-programme appelé. Les paramètres formels sont des variables locales dont les valeurs après exécution des instructions de l'appelé ne peuvent pas être récupérées par l'appelant. Ce mode assure donc une protection des variables de l'appelant transmises par valeur. Elles gardent les mêmes valeurs au niveau de l'appel du sous-programme et juste après l'exécution de cet appel.

Dans notre exemple, l'appel de la fonction Moyenne (Coef1, Note1, Coef2, Note2), les paramètres effectifs Coef1, Note1, Coef2 et Note2 sont transmis par valeur. Les paramètres formels correspondants c1, n1, c2 et n2 seront tout simplement "initialisées" par les quatre valeurs Coef1, Note1, Coef2 et Note2 au moment de l'appel de cette fonction.

Remarque :

Dans la partie instruction du sous-programme appelé (exemple : fonction Moyenne), les paramètres formels transmis par valeur (exemple : c1, n1, c2 et n2) ne doivent en aucun cas se trouver à gauche du symbole d'une instruction d'affectation. Même si par erreur la valeur du paramètre formel transmis par valeur (exemple : c1, n1, c2 ou n2) est modifiée dans l'appelé, au retour après exécution de celui-ci, le paramètre effectif correspondant (exemple : Coef1, Note1, Coef2 ou Note2) garde la même valeur originale transmise lors de l'appel. En effet on ne manipule au niveau de l'appelé qu'une copie locale.

II.4 Activation d'une fonction et retour du résultat de la fonction

II.4.1 Activation d'une fonction

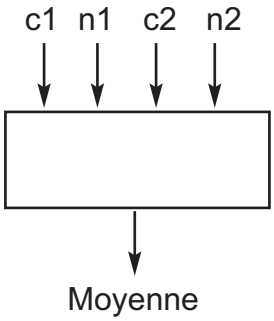
L'activation d'une fonction est provoquée par l'exécution d'une instruction d'appel. Dans notre exemple, Moyenne (Coef1, Note1, Coef2, Note2) constitue l'appel à la fonction Moyenne.

L'activation de la fonction après passage des paramètres se déroule dans l'ordre suivant :

1. Déclarations des objets locaux ; ce qui provoque leur création (et notamment les places mémoires nécessaires aux différentes variables locales).
2. Exécution des instructions qui composent le corps de la fonction.
3. Sortie de la fonction provoquée une fois que le corps est exécuté et que l'on atteint l'instruction Fin. Il y a alors retour à l'appelant qui continue l'exécution de l'instruction en cours ou l'instruction qui suit l'instruction d'appel.

II.4.2 Retour du résultat de la fonction

Une fonction possède un type, c'est celui du résultat qu'elle calcule. Dans notre exemple, DEF FN Moyenne (c1, n1, c2,n2 : Réel) : **Réel**, la fonction Moyenne rend un résultat du type Réel.



Donc, il doit y avoir nécessairement dans la partie instruction de la fonction au moins une affectation explicite ayant l'identificateur de la fonction à gauche du symbole d'affectation.
Dans notre exemple, ceci est illustré par :

Moyenne ← Total / (Coef1+ Coef2).

La dernière valeur affectée à cet identificateur lors de l'appel de la fonction est le résultat de l'évaluation de la fonction.
Une fonction peut être appelée à l'intérieur d'une expression (exemple y ←3*f(x)). L'appel de la fonction ne pourra pas se trouver à gauche du symbole de l'affectation dans le sous-programme ou programme appelant !
Une fonction calcule une valeur qui, retournée au sous-programme ou programme appelant, est immédiatement utilisable. Son nom est le seul paramètre résultat et tous les paramètres explicitement transmis sont en principe des données.

II.5 Exemple d'exécution commentée du programme Calcul_Moy

```
Affichage :   Entrer le 1er coefficient :
Lecture   :   3       {  Coef1 }
Affichage :   Entrer le 2ème Coefficient :
Lecture   :   2       {  Coef2 }
              Lecture   :        2     {  n=2 élèves - 2 itérations }
Lecture   :   11.5    {  Note1 à la 1ère itération }
Lecture   :   10      {  Note2 à la 1ère itération }
                Moy :=          exécution de la fonction Moyenne
                                      N1 initialisée à 11.5
                                      N2 initialisée à 10
                                      Total := 3*11.5+2*10)    {* Total :=54.5 *}
                                       Moyenne := 54.5/(3+2)    {* Moyenne :=10.90
                                            Le résultat de Moyenne est 10.90*}
                        Moy :=10.90     {  valeur résultat rendu par moyenne }
```

```
Affichage :   Élève 1 a une Moyenne = 10.90
Lecture :     13      { Note1 à la 2ème itération }
Lecture :     16      { Note2 à la 2ème itération }
              Moy :=        exécution de la fonction Moyenne
                                  N1 initialisée à 13
                                  N2 initialisée à 16
                                  Total := 3*13 + 2*16        {* Total :=71 *}
                                  Moyenne := 71/(3+2)         {* Moyenne :=14.20
                            Le résultat de Moyenne est 14.20}
                 Moy :=14.20 {  valeur résultat rendu par moyenne }

Affichage :   Élève 2 a une Moyenne = 14.20
```

Retenons

Les objets déclarés dans un programme ou dans un sous-programme sont dits locaux à celui-ci.
Les sous-programmes peuvent manipuler des objets globaux ; mais, ils devront de préférence communiquer à travers des paramètres.

Exercices

Exercice 1

1. Quelle est la différence entre paramètre formel et paramètre effectif ?
2. Quels sont les sous-programmes qui peuvent accéder à un objet déclaré dans un sous-programme F ?
3. Quel est le mode de passage des paramètres utilisé pour les fonctions ?

Exercice 2

1. Ecrire un programme Pascal qui permet de calculer le minimum et le maximum de deux entiers en utilisant deux fonctions MIN et MAX.
2. Donner le bilan des objets utilisés dans ce programme :
   - Les objets globaux
   - Les objets locaux (par fonction)
   - les paramètres formels
   - les paramètres effectifs

Exercice 3

Soit le programme Pascal suivant :

```
PROGRAM Suite ;
VAR n1,n2 : INTEGER;
    Un,Sn: REAL ;

FUNCTION Factoriel ( k : INTEGER ): INTEGER ;
  VAR   i,f: INTEGER ;
  BEGIN
   f := 1 ;
   FOR  i := 1 TO k  DO
     BEGIN
       f := f * i ;
     END ;
   Factoriel := f ;
END ;

FUNCTION nieme_terme (l : INTEGER) : REAL ;
  VAR i,S : INTEGER ;
    BEGIN
      S := 0 ;
    FOR  i := 1 TO l  DO
        BEGIN
           S := S + (2*i + 1) ;
        END ;
    nieme_terme := S / Factoriel(l) ;
END ;
```

```
FUNCTION somme_suites (m : INTEGER) : REAL ;
   VAR i : INTEGER ;
       S : REAL ;
   BEGIN
       S := 0 ;
       FOR  i := 1 TO m DO
            BEGIN
               S := S + nieme_terme(i) ;
            END ;
       somme_suites := S ;
   END ;

BEGIN  { PROGRAMME PINCIPAL}
  WRITELN(' Donner un entier : ') ;
  READLN (n1) ;
  Un := nieme_terme(n1) ;
  WRITELN('Donner un entier : ') ;
  READLN (n2) ;
  Sn := somme_suites(n2) ;
  WRITELN('Le ',n1,'ieme terme de la suite est : ', Un) ;
  WRITELN('La somme du ',n2, 'premiers termes de la suite
           est : ',Sn) ;
END.
```

Donner les différents niveaux des sous-programmes utilisés en utilisant un arbre. Puis donner les règles d'accès aux objets par les différents sous-programmes (accessible ou inaccessible).

Exercice 4

Soit la fonction suivante :

0) DEF FN F(x:entier):entier
1) $x \leftarrow x + 3$
2) $F \leftarrow x$
3) Fin F

Écrire un programme en Pascal contenant cette fonction. Dans votre programme principal vous définissez une variable y initialisée à 0 ; vous afficherez cette variable avant d'appeler la fonction **F** ainsi qu'après.
Est-ce que la valeur de la variable y affichée après appel de la fonction F a changé? Pourquoi ?

Leçon 4

Les procedures

Objectifs spécifiques

- Comprendre l'intérêt des sous-programmes de type procédure
- Savoir déclarer, programmer et utiliser les procédures

Plan de la leçon

Leçon 4

Les procedures

Le bon sens est la chose au monde la mieux partagée : car chacun pense en être si bien pourvu, que ceux mêmes qui sont les plus difficiles à contenter en toute chose n'ont point coutume d'en désirer plus qu'ils en ont.

DESCARTES

I. Vocabulaire et syntaxe

I.1 Définition

Les procédures sont des sous-programmes qui peuvent avoir zéro, un ou plusieurs résultats.
Une procédure permet de résoudre un sous-problème.

I.2 Vocabulaire et syntaxe

L'appel de la procédure doit se trouver dans une instruction d'appel et ne peut pas être dans une expression comme le cas des fonctions.
Dans l'analyse de l'algorithme nous écrivons :
PROC identificateur_procedure(paramètre1, paramètre2, paramètren)

Activité 1

Écrire un programme permettant de saisir le nom et la note de n candidats. Il doit ensuite appeler une procédure Concours qui affiche l'admission éventuelle de cette personne au concours. On fera la saisie de n noms et notes.

Analyse

	Nom : Recrutements	
S	**L.D.E.**	**O.U.**
	Résultat = Les_recrutements	
1	Les_recrutements = Ecrire("Donner le Nbre de personnes:") n = Donnée	n
	Pour i de 1 à n **Faire**	i
	Nom = Donnée	Nom
	Note = Donnée	Note
	PROC Concours(Nom,Note)	Concours
	FinPour	
2	**Fin Recrutements**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
n	Entier	Nombre de candidats au concours
i	Entier	Compteur
Nom	Chaîne	Nom d'une personne donnée
Note	Réel	Note obtenue au concours
Concours	Procédure	Procédure permettant d'afficher l'admission au concours d'une personne donnée

Algorithme

```
0) début  Recrutements
1) Ecrire (" donner le nombre de personnes  : ")
   Lire (n)
     Pour i de 1 à n Faire
              Lire (Nom)
              Lire (Note)
              PROC Concours(Nom,Note)
     FinPour
2) Fin Recrutements.
```

Analyse de la procédure Concours

	DEF PROC Concours (Lenom : chaîne , Note_eval : Réel) ;	
S	**L.D.E.**	**O.U.**
2 1 3	**Résultat** =Ecrire ("M. ", Lenom , message) [message]= [ ]**Si** (Note_eval > 14) **Alors** message←"vous êtes admis au concours " **Sinon** message←"vous n'êtes pas admis au concours " **Finsi** **Fin Concours**	Lenom Note_eval message

Algorithme

```
0) DEF PROC Concours ( Lenom : Chaîne , Note_eval : Réel )
1) Si  ( Note_eval > 14 ) Alors  message←" vous êtes admis au concours "
   Sinon message←" vous n'êtes pas admis au concours"
  Finsi
2) Écrire (" M. ", Lenom , message )
3) Fin Concours
```

Tableau de déclaration des objets

Objet	Type/nature	Rôle
Message	Chaîne	Message à adresser au candidat

Traduction en Pascal

```
PROGRAM Recrutements;
USES WINCRT;
VAR   n ,i : INTEGER ;
      Nom : STRING ;
      Note : REAL ;

PROCEDURE  Concours ( Lenom:STRING,Note_eval:REAL );
VAR message : STRING ;
BEGIN
   IF (Note_eval > 14) THEN
          Message := ' vous êtes admis au concours '
   ELSE
         Message := ' vous n''êtes pas admis auconcours ' ;

  WRITELN('M. ',Lenom, Message );
END ;

BEGIN
  WRITELN ('donner le nbre de personnes :') ;
  READLN(n) ;

  FOR  i := 1 TO n  DO
   BEGIN
        WRITE('Entrer le nom : ');READLN(Nom) ;
        WRITE('Entrer la note : ');READLN(Note) ;
        Concours(Nom, Note) ;
   END ;
END.
```

Un cas d'exécution

```
Donner le nbre de personnes :
2
Entrer le nom : JAMEL
Entrer la note : 17
M. JAMEL vous êtes admis au concours
Entrer le nom : ALI
Entrer la note : 9
M. ALI vous n'êtes pas admis au concours
```

Dans l'activité 1, l'appel est réalisé par l'instruction : PROC Concours (Nom,Note).

Les définitions des objets locaux et globaux pour une procédure suivent les mêmes règles que celles édictées pour les fonctions.

L'entête de la procédure Concours est

DEF PROC Concours (Lenom : Chaîne , Note_eval : Réel).

Lenom et Note_eval sont deux paramètres formels avec un mode de transmission par valeur que nous avons traité lors de l'étude des fonctions. Nous allons définir dans le paragraphe suivant le mode de transmission par variable.

II. Mode de transmission par variable

Activité 2

Soit T un tableau de n caractères (2≤n≤20). On se propose d'écrire un programme qui saisit n et T, ensuite il appelle une procédure Occurrence qui cherche le nombre de caractères alphabétiques et le nombre de chiffres dans le tableau T.

Analyse

	Nom = Cherche_nbr	
S	**L.D.E.**	**O.U.**
3	**Résultat =** Ecrire ("Alphabétique = ", Nbr_A, "Chiffre = " , Nbr_C)	Nbr_A
	(Nbr_A, Nbr_C) = PROC Occurence (T, Nbr_A, Nbr_C)	Nbr_C
2	T = [n = Donnée("Entrer le nombre d'éléments")]	T
	Pour i de 1 à n **Faire**	n
1	T[i] = Donnée(" Entrer un élément")	i
	FinPour	
4	**Fin cherche_nbr**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
Nbr_A	Entier	Paramètre effectif résultat, nombre de caractères alphabétiques
Nbr_C	Entier	Paramètre effectif résultat, nombre de chiffres
T	Tableau	Tableau des caractères
n	Entier	Nombre d'éléments de T
i	Entier	Compteur

Algorithme

0) Début Cherche_nbr
1) Ecrire("Entrer le nombre d'éléments"), (Lire (n)
Pour i de 1 à n Faire
Ecrire(" Entrer un élément"), Lire (T(i))
FinPour
2) PROC Occurence (T, Nbr_A, Nbr_C)
3) Ecrire ("Alphabétique = ", Nbr_A, "Consonne = " , Nbr_C)
4) Fin Chercher_nbr.

Analyse de la procédure Concours

DEF PROC Occurrence **(TE : tableau de n caractères, VAR A : Entier, VAR C : Entier)**

S	L.D.E.	O.U.
1	**Résultat** = OCCURRENCE OCCURRENCE = [A ← 0 ; C ← 0] **POUR** i de 1 à n **Faire** **[ ] Si** MAJUS(TE[i]) Dans ["A".."Z"] **Alors** A ← A +1 **Sinon si** (TE[i]) Dans ["1".."9"] **Alors** C ← C +1 **FinSi** **Fin POUR**	A C i TE n
2	**Fin Occurrence**	

Remarque :

Dans l'analyse de la procédure Occurrence, l'objet utilisé n est un objet global. Comme nous le verrons par la suite, n sera déclaré dans le programme appelant qui est dans ce cas le programme principal.

Tableau de déclaration des objets

Objet	Type/nature	Rôle
i	Entier	Compteur

Algorithme

```
0) DEFPROC Occurrence ( TE : tableau de n entiers, VAR A : entier, VAR C : entier )
1) A ← 0
   C ← 0
   POUR I de 1 à n Faire
            Si   MAJUS(TE[ i ]) Dans ["A".."Z"]
                        Alors
                          A  ← A +1
            Sinon si   (TE[ i ]) Dans ["1".."9"]
                        Alors
                          C  ← C +1
           FinSi
   Fin POUR

2) Fin Occurrence
```

Traduction en Pascal

```
PROGRAM Cherche_nbr;
USES WINCRT ;
TYPE Typtab=ARRAY [1..30] OF CHAR ;
VAR
   T : Typtab ;
   n,i,Nbr_A,Nbr_C : INTEGER;

PROCEDURE Occurrence(TE:Typtab; VAR A:INTEGER; VAR C:INTEGER);
BEGIN
   A := 0 ;
   C := 0 ;
      FOR i := 1 TO n  DO
                  BEGIN
                     IF  UPCASE(TE[i])  IN  ['A'..'Z']
                                        THEN
                                              A := A +1
                     ELSE IF    (TE[i]) IN ['1'..'9']
                                        THEN
                                              C  := C +1;
                  END;
END ;
```

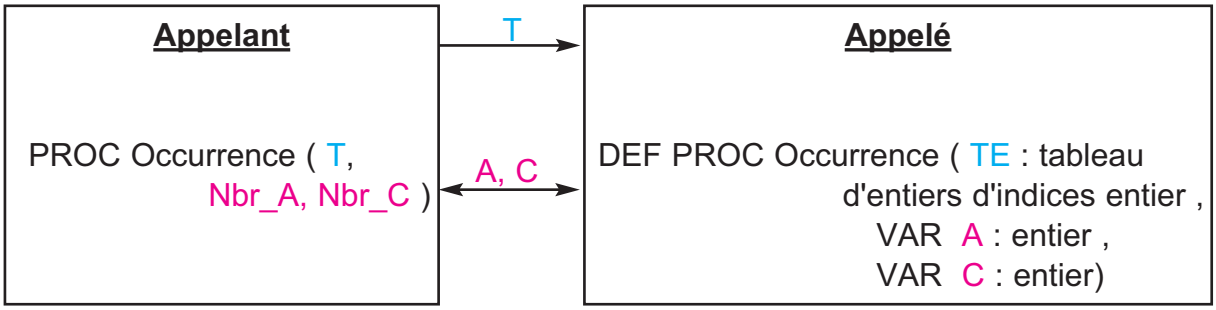
```
BEGIN
   A := 0 ;
   C := 0 ;
      FOR i := 1 TO n  DO
                  BEGIN
                     IF  UPCASE(TE[i]) IN ['A'..'Z'] THEN
                                          A := A +1
                     ELSE IF    (TE[i]) IN ['1'..'9'] THEN
                                           C  := C +1;
                  END;
END ;

BEGIN
WRITELN('Entrer le nombre d''éléments du tableau');READLN( n );
   FOR    i := 1  TO  n  DO
          BEGIN
            WRITE('Entrer un élément: ');
            READLN (T[i]);
          END;
   Occurrence ( T,Nbr_A,Nbr_C);
   WRITELN ('Nombre de caractères alphabétiques est: ', Nbr_A);
   WRITELN('Nombre de Chiffres est : ' , Nbr_C);
END.
```

Un cas d'exécution

```
Entrer le nombre d'éléments du tableau
10
Entrer un élément:  *
Entrer un élément:  R
Entrer un élément:  A
Entrer un élément:  M
Entrer un élément:  5
Entrer un élément:  1
Entrer un élément:  2
Entrer un élément:  M
Entrer un élément:  0
Entrer un élément:  *
Nombre de caractères alphabétiques est: 5
Nombre de Chiffres est : 3
```

Dans l'activité 2, l'appel est réalisé par l'instruction :
PROC Occurrence (T, Nbr_A, Nbr_C).

L'entête de la procédure Occurrence est
DEF PROC Occurrence (TE : tableau d'entiers d'indices entier ,
VAR A : entier , VAR C : entier).



TE est le paramètre formel avec un mode de transmission par valeur. A et C sont des paramètres formels avec un mode de transmission par variable. C'est ce que nous allons définir dans ce qui suit.

Le paramètre effectif est une **variable** qui peut être une variable simple ou structurée (tableau), ou élément d'un tableau. Dans l'entête de la procédure, on doit faire précéder les paramètres formels transmis par variable par le mot-clé **VAR**. S'ils sont d'un même type, on pourra les regrouper après un même VAR.

Les paramètres transmis par variable font communiquer l'information dans les deux sens :
A l'appel, l'appelant transmet une ou plusieurs valeurs à l'appelé ; de même l'appelé rend un ou plusieurs résultats à l'appelant au moment du retour après activation de la procédure.

Il faut donc utiliser ce mode de transmission par variable chaque fois que le paramètre représente un résultat calculé par le sous-programme appelé et transmis au sous-programme (ou programme) appelant.

Dans ce mode de passage, le paramètre effectif et le paramètre formel correspondant deviennent synonymes et ont la même adresse (la même référence en mémoire). Cela implique que toute modification d'un paramètre formel dans le sous-programme appelé (dans la partie instruction) entraîne au moment du retour à l'appelant une modification de la valeur du paramètre effectif correspondant.

Dans l'instruction d'appel PROC Occurrence (T, Nbr_A, Nbr_C) de l'exemple précédent, les variables Nbr_A et Nbr_C sont des paramètres effectifs recevant en retour un résultat de la part de la procédure Occurrence au retour une fois son activation terminée. Le mode de transmission est alors nécessairement par variable. Dans l'entête de la procédure Occurrence, les paramètres formels correspondant A et C sont précédés par le mot-clé **VAR** .

Activité 3

Ecrire un programme qui répète n fois la saisie de deux valeurs x et y, affiche ces valeurs puis appelle un sous-programme permettant de permuter ces deux valeurs et il affiche à chaque fois les deux valeurs permutées.

Analyse

	Nom : Lire_permut	
S	**L.D.E.**	**O.U.**
1	**Résultat** = Lire_permuter Lire_ permuter =[n = Donnée("Nombre de couples à saisir >=1")] **Pour** i de 1 à n **Faire** x =Donnée("Entrer la première valeur") y =Donnée("Entrer la deuxième valeur") Ecrire ("x =", x, "y =", y) PROC Permut (x, y) Écrire ("x =", x, "y =", y) **FinPour**	n i x y
2	**Fin Lire_permut**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
n	Entier	Nombre de couples à saisir >=1
i	Entier	Compteur
x	Réel	1ère valeur
y	Réel	2ème valeur

Algorithme

```
0) Début  Lire_Permut
1) Ecrire("Nombre de couples à saisir >=1"), Lire (n)
    Pour i de 1 à n Faire
        Ecrire("Entrer la première valeur"), Lire (x)
        Ecrire("Entrer la deuxième valeur "), Lire (y)
        Ecrire ("x =", x, "y =", y )
        PROC Permut (x, y)
        Ecrire ("x =", x, "y =", y )
    FinPour
2) Fin Lire_Permut
```

Analyse de la procédure Permut

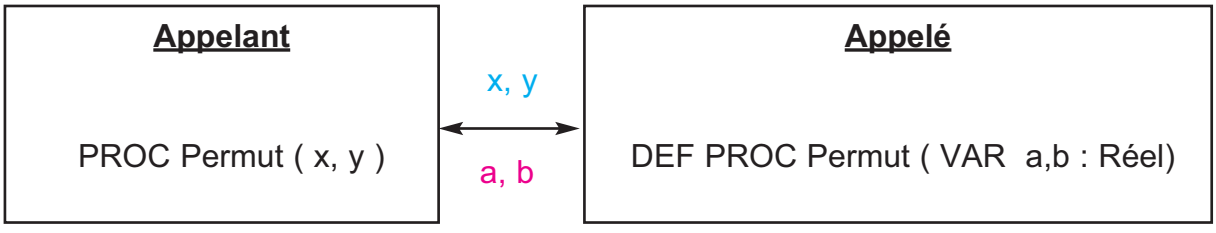
	DEF PROC Permut (VAR a, b : réel)	
S	**L.D.E.**	**O.U.**
3 1 2 4	**Résultat** = (a, b) a ← aux aux ← b b ← a **Fin Permut**	a b aux

Tableau des déclarations des objets locaux

Objet	Type/nature	Rôle
aux	Réel	Variable intermédiaire

Algorithme

0) DEF PROC Permut (VAR a, b : Réel)
1) aux ← b
2) b ← a
3) a ← aux
4) Fin Permut



Remarque :

Dans cet exemple, les paramètres formels a et b sont initialisés au niveau de l'appel par les valeurs de x et de y puis une fois les variables a et b permutées, au retour, leurs valeurs sont communiquées en tant que résultats à l'appelant. Les variables x et y reçoivent alors les nouvelles valeurs résultats. Ces paramètres sont des paramètres Données / Résultats. Cet exemple nous montre l'intérêt de l'utilisation du mode de passage par variable.

Traduction en Pascal

```
PROGRAM   LIRE_PERMUT ;
USES WINCRT;
VAR      n,i : INTEGER ;
         x,y : REAL ;
PROCEDURE Permute( VAR a, b : REAL );
VAR aux : REAL ;
BEGIN
   aux := b ;
   b := a ;
   a := aux ;
END ;
BEGIN
  WRITE('Nombre de couples à saisir >=1 : ') ;
  READLN(n) ;
  FOR    i := 1 TO   n   DO
         BEGIN

              WRITE('Entrer la première valeur ');
              READLN (x);
              WRITE('Entrer la deuxième valeur ');
              READLN (y);
              WRITELN ('x = ', x:5 :2, '   y = ', y:5 :2);
              Permute (x, y ) ;
              WRITELN ('x = ', x:5 :2, '   y = ', y:5 :2);
         END ;
END.
```

Un cas d'exécution

```
Nombre de couples à saisir >=1 :  3
Entrer la première valeur 1
Entrer la deuxième valeur 7
x = 1.00  y =  7.00
x = 7.00  y =  1.00
Entrer la première valeur 6.6
Entrer la deuxième valeur 9
x = 6.60  y =  9.00
x = 9.00  y =  6.60
Entrer la première valeur 13
Entrer la deuxième valeur 25
x =13.00  y = 25.00
x =25.00  y = 13.00
```

Remarque :

Il est possible de déclarer un sous-programme **sans paramètre**. La communication avec l'appelant pourrait se produire grâce aux ressources communes (objets globaux) partagées entre l'appelé et l'appelant.

Exemple de procédure sans paramètre :

```
PROGRAM Affiche ;
USES WINCRT ;

PROCEDURE Affiche_message;
VAR  message: STRING;
BEGIN
  WRITELN('Procédure sans paramètre ');
  WRITELN('Cette procédure a pour rôle d''afficher un
                                          message saisi ');
  WRITELN('à l''écran à partir de la position x=10, y=10');
  WRITELN('Donner votre message :  ') ;
  READLN(message) ;
  CLRSCR ;
  GOTOXY(10,10) ;
  WRITELN(message) ;
END.

BEGIN
  Affiche_message;
END.
```

Un cas d'exécution

```
Procédure sans paramètre
Cette procédure a pour rôle d'afficher un message saisi
à l'écran à partir de la position x=10, y=10
Donner votre message :
INFORMATIQUE_


         INFORMATIQUE
```

III. Problème corrigé

Soit T un tableau de n caractères numériques avec n=20. Écrire un programme permettant de crypter les données figurant dans le tableau T comme suit :

1. Convertir les éléments de T compris entre deux positions P1 et P2 en leurs représentations en code ASCII.
2. Permuter les chiffres des unités avec ceux des dizaines du code ASCII puis déterminer le caractère correspondant à ce nouveau code ASCII.
3. Faites une permutation de façon circulaire vers la droite d'ordre 1 sur le contenu du tableau T.
   On donne 1≤ P1< P2 ≤ 20.

Analyse

	DEF PROC CRYPT_T(VAR a, b : réel)	
S	**L.D.E.**	**O.U.**
	Résultat = T	n
6	PROC AfficherT(n,T)	T
5	PROC Permut_circulaire(n,T)	P1
4	PROC Crypt(T,P1,P2)	P2
3	PROC Saisir_pos(P1,P2)	AfficherT
2	PROC AfficherT(n,T)	Permut_circulaire
1	PROC RemplirT(n,T)	Saisir_pos
7	**Fin CRYPT_T**	RemplirT

Tableau de déclaration des objets

Objet	Type/nature	Rôle
n	Entier	Taille du tableau
T	Tab	Tableau de n caractères
P1	Entier	Première position du tableau
P2	Entier	Deuxième position du tableau
AfficherT	Procédure	Affiche le contenu du tableau
Permut_circulaire	Procédure	Permute de façon circulaire le tableau
Saisir_pos	Procédure	Saisit de deux positions P1 et P2
RemplirT	Procédure	Remplissage du tableau

	DEF PROC RemplirT(var n:Entier ; var A:TAB)	
S	**L.D.E.**	**O.U.**
	Résultat = A	a
2	A = [] **Pour I** de 1 à n **Faire**	b
	Répéter	aux
	Ecrire(" Entrer élément ",I, " : ")	
	Lire(A[I])	
	Jusqu'à ORD(A[i]) **DANS** [48..57]	
	FinPour	
1	n = [] **Répéter**	
	Ecrire(" Entrer la taille du tableau: ")	
	Lire(n)	
	Jusqu'à n=20	
3	**Fin RemplirT**	

Objet	Type/nature	Rôle
I	Entier	Compteur

	DEF PROC Saisir_pos(var P1,P2:Entier)	
S	**L.D.E.**	**O.U.**
	Résultat = (P1,P2)	
1	(P1,P2) = [] **Répéter**	
	Ecrire (" Entrer la position P1 : ")	
	Lire (P1)	
	Ecrire (" Entrer la position P2 : ")	
	Lire (P2)	
	Jusqu'à ((P1>=1)**ET**(P1<P2)**ET(**P2<=n))	
2	**Fin Saisir_pos**	

	DEF PROC Crypt(var T: TAB ; P1,P2:Entier)	
S	**L.D.E.**	**O.U.**
	Résultat = T	ORD1
1	**T** = [] **Pour I** de P1 à P2 **Répéter**	ORD2
	ORD1 ← ORD(T[I])	Inverse
	ORD2 ← FN Inverse (ORD1)	
	T[I] ← CHR(ORD2)	
	FinPour	
2	**Fin Crypt**	

Objet	Type/nature	Rôle
ORD1	Entier	Contient l'ordre du caractère
ORD2	Entier	Contient l'ordre du caractère inversé
Inverse	Fonction	Inverse un chiffre

DEF FN Inverse (X:**Entier**):**Entier**

S	L.D.E.	O.U.
4	**Résultat** = Inverse	U
3	Inverse:=U*10+D	D
2	U:=X **MOD** 10	
1	D:=X **DIV** 10	
5	**Fin Inverse**	

Objet	Type/nature	Rôle
U	Entier	Contient le chiffre d'unité
D	Entier	Contient le chiffre de dizaine

DEF PROC Permut_circulaire(n:**Entier** ; **var** T:TAB)

S	L.D.E.	O.U.
	Résultat = T	Temp
2	T[1] ← Temp	i
1	T = [Temp ← T[n]] **Pour** I de n à 2 (pas = -1) **Faire**	
	T[i] ← T[i-1]	
	FinPour	
3	**Fin Permut_circulaire**	

Objet	Type/nature	Rôle
Temp	Entier	Sauvegarde d'un élément du tableau
i	Entier	Compteur

DEF PROC AfficherT(n:**Entier** ; T:TAB)

S	L.D.E.	O.U.
	Résultat = T	i
1	T = [ ] **Pour i de** 1 **à** n **Répéter**	
	Ecrire(T[i]," ")	
2	**Fin Pour**	
	AfficherT	

Objet	Type/nature	Rôle
i	Entier	Compteur

Algorithmes du programme principal

```
0)Début CRYPT_T
1)PROC RemplirT(n,t)
2)PROC AfficherT(n,T)
3)PROC Saisir_pos(P1,P2)
4)PROC Crypt(P1,P2,T)
5)PROC Permut_circulaire(T,n)
6)PROC AfficherT(n,T)
7)Fin CRYPT_T
```

Algorithmes du sous programme RemplirT

```
0)DEFPROC RemplirT(var n:Entier ; var A:TAB)
1)Répéter
      Ecrire(" Entrer la taille du tableau: ")
      Lire(n)
  Jusqu'à n=20

2)Pour i de 1 à n Répéter
     Répéter
       Ecrire(" Entrer élément ",i, " : ")
       Lire(A[I])
     Jusqu'à ORD(A[i]) DANS [48..57]
  FinPour
3)Fin RemplirT
```

Algorithmes du sous programme Saisir_pos

```
0)DEFPROC Saisir_pos(var P1,P2:Entier)
1)Répéter
      Ecrire (" Entrer la position P1 : ")
      Lire (P1)
      Ecrire (" Entrer la position P2 : ")
      Lire (P2)
  Jusqu'à (P1>=1)AND(P1<P2)AND(P2<=n)
2)Fin Saisir_pos
```

Algorithmes du sous programme Crypt

0)DEFPROC Crypt(var T: TAB ; P1,P2:Entier)
1)Pour i de P1 à p2 Répéter
ORD1 ← ORD(T[i])
ORD2 ← FN Inverse (ORD1)
T[i] ← CHR(ORD2)
FinPour
2)Fin Crypt

Algorithmes du sous programme Inverse

0)DEFFN Inverse (X:Entier):Entier
1)D:=X DIV 10
2)U:=X MOD 10
*3)Inverse:=U*10+D*
4)Fin Inverse

Algorithmes du sous programme Permut_circulaire

0)DEFPROC Permut_circulaire(n:Entier ; var T:TAB)
1)Temp ← T[n];
FOR I ← n DOWNTO 2 Répéter
T[i] ← T[i-1]
FinPour
2)T[1] ← temp
3)Fin Permut_circulaire

Algorithmes du sous programme AfficherT

0)DEFPROC AfficherT(n:Entier ; T:TAB)
1)Pour i de 1 à n Répéter
Ecrire(T[i]," ")
2)Fin AfficherT

Traduction en Pascal

```
PROGRAM    CRYPT_T;
USES WINCRT;
TYPE
  TAB = ARRAY [1..50] OF CHAR;
```

```
VAR
  T: TAB;
  P1,P2,n: INTEGER;
  i: INTEGER;

 PROCEDURE RemplirT(var n:INTEGER ; var A:TAB);
  BEGIN
    REPEAT
       WRITE('Entrer la taille du tableau: ');
       READLN(n);
    UNTIL n=20;

    FOR i := 1 TO n DO
     BEGIN
       REPEAT
         WRITE ('Entrer élément ',i,' : ');
         READLN (A[i]);
       UNTIL ORD(A[i]) IN [48..57];
     END;
  END;

 PROCEDURE Saisir_pos(var P1,P2:INTEGER);
  BEGIN
    REPEAT
      WRITE ('Entrer la position P1 : ');
      READLN (P1);
      WRITE ('Entrer la position P2 : ');
      READLN (P2);
    UNTIL ((P1>=1)AND(P1<P2)AND(P2<=n));
  END;

PROCEDURE Crypt(var T: TAB;P1,P2:INTEGER);
    VAR
      ORD1,ORD2,I: INTEGER;

    FUNCTION Inverse (X:INTEGER):INTEGER;
    VAR
      D,U:INTEGER;
    BEGIN
       D:=X DIV 10;
       U:=X MOD 10;
       Inverse:=U*10+D;
    END;
```

```
    BEGIN

     FOR i := P1 TO p2 DO
       BEGIN
          ORD1:=ORD(T[i]);
          ORD2:=Inverse (ORD1);
          T[i]:=CHR(ORD2);
       END;

   END;

  PROCEDURE Permut_circulaire(n:INTEGER ; var T:TAB );
    VAR
      i:INTEGER;
      temp:CHAR;
   BEGIN
   temp:=T[n];
   FOR i := n DOWNTO 2 DO
         T[i]:=T[i-1];
   T[1]:=temp;

  END;

  PROCEDURE AfficherT(n:INTEGER ; T:TAB);
   BEGIN
      FOR i := 1 TO n DO
      WRITE (T[i],'  ');
      WRITELN;
   END;

BEGIN
 RemplirT(n,T);
 AfficherT(n,T);
 Saisir_pos(P1,P2);
 Crypt(T,P1,P2);
 Permut_circulaire(n,T);
 AfficherT(n,T);
END.
```

Un cas d'exécution

```
Entrer élément 1 : 1
Entrer élément 2 : 2
Entrer élément 3 : 3
Entrer élément 4 : 4
Entrer élément 5 : 9
Entrer élément 6 : 8
Entrer élément 7 : 7
Entrer élément 8 : 6
Entrer élément 9 : 5
Entrer élément 10 : 3
Entrer élément 11 : 7
Entrer élément 12 : 8
Entrer élément 13 : 9
Entrer élément 14 : 9
Entrer élément 15 : 0
Entrer élément 16 : 6
Entrer élément 17 : 6
Entrer élément 18 : 5
Entrer élément 19 : 4
Entrer élément 20 : 2
1  2  3  4  9  8  7  6  5  3  7  8  9  9  0  6  6  5  4  2
Entrer la position P1 : 5
Entrer la position P2 : 11
2  1  2  3  4  K  A  7  -  #     7  8  9  9  0  6  6  5  4
```

Retenons

Dans le cas où une procédure retourne un résultat, vous devez choisir le mode de transmission par variable pour le paramètre résultat en question. Vous devez faire précéder le paramètre formel par le mot clé VAR.

Exercices

Exercice 1

On veut écrire un programme qui détermine le nombre de mots dans une phrase. La phrase commence obligatoirement par une lettre. Devant chaque mot d'une phrase il y a un caractère blanc et les signes de ponctuation sont collées aux mots les précédant.

Questions :

1. Ecrire les modules effectuant chacun les traitements décrits ci-dessous.

- ✓ Ecrire une procédure nommée LECTURE qui lit une phrase.
- ✓ Ecrire une procédure nommée COMPTE qui compte et affiche le nombre de mots dans la phrase.

2. Ecrire l'algorithme du programme principal .

3. Traduire cet algorithme en Pascal.

Exercice 2

Écrire un procédure intitulé COMB_NBRE qui permet d'afficher toutes les combinaisons d'un nombre qui s'écrit sous la forme de 3 chiffres.

Questions :

1. Saisir n dans le programme principal

2. Appeler la procédure COMB_NBRE permettant de chercher et d'afficher les différentes combinaisons.

Exercice 3

On veut écrire un programme qui fait l'inversion des éléments d'un vecteur T à N entiers (N est compris entre 1 et 100).

Exemple :	**Donnée**	-10	15	12	6	-4	5	18
	Résultat	18	5	-4	6	12	15	-10

Questions :

1. Analyser chacun de ces modules décrits ci-dessous.

- ✓ Ecrire la procédure LECTURE (VAR N ; VAR T) qui range N entiers dans un tableau T.
- ✓ Ecrire la procédure INVERSE (VAR X , Y) qui permet de permuter deux entiers X et Y donnés.
- ✓ Ecrire la procédure INVERSE_TOUT (VAR T) qui inverse tous les éléments du vecteur T en utilisant la procédure INVERSE.
- ✓ Traduire cet algorithme en Pascal.
- ✓ Ecrire la procédure RESULTAT (T) qui affiche le vecteur T résultat.
- ✓ Ecrire l'analyse du programme principal.

2. En déduire les algorithmes relatifs à chacun de ces modules ainsi que celui du programme principal.

3. Traduire ces algorithmes en Pascal.

Exercice 4

Un nombre premier est un entier divisible uniquement par 1 et par lui même.

Écrire un programme permettant d'afficher les n nombres premiers (n est un entier strictement positif donné).

Exercice 5

T étant un tableau d'entiers contenant au maximum 20 éléments. Écrire un programme principal qui permet de :

- ✓ saisir N éléments dans T
- ✓ Saisir la valeur de l'élément x à insérer
- ✓ Saisir la position d'insertion p dans le tableau
- ✓ d'appeler une procédure permettant d'insérer l'élément x à la position p dans le tableau.

Exercice 6

Soit la procédure P :

0) DEF PROC P(x:entier ; VAR F:entier)
1) x ← x + 3
2) F← x
3) Fin P

Questions :

1. Quel est le mode de passage du paramètre x ?
2. Quel est le mode de passage du paramètre F ?
3. Quels sont les avantages de proposer les deux types de passage de paramètre par adresse et par valeur au sein d'un même langage.

Exercice 7

Soit le programme Pascal suivant qui comporte plusieurs sous programmes de différents niveaux d'imbrications.

```
program X ;
  var A;
   procedure Y ;
      var B ;
      procedure Y1 ;
        var C ;
       begin
         C := B + A
       end ;
      procedure Y2 ;
        var B ;
      begin
          Y1
       end ;
    begin
      Y2
    end ;
  begin
    Y
  end .
```

Question :

Compléter le tableau ci-dessous pour déterminer la nature (variable ou procédure) et les niveaux des différents objets utilisés.

Objet	Nature	NIVEAU
C		
Y1		
B		
Y		
A		
Y2		
X		

Exercice 8

Soit l'algorithme suivant :

0) DEF PROC SomCar (X1 : entier, X2 : entier, S : entier)
1) X1← X1 * X1
2) X2 ← X2 * X2
3) S ← X1 + X2
4) Fin SomCar

0) Début **Programme principal**
1) X ← 3
2)Y ← 4
3) Z ← 0
4) SomCar(X, Y, Z)
5) Ecrire (X,Y, Z)
Fin du programme principal

Questions :

1. Trouver le résultat fourni par l'algorithme ci-dessus
2. Que faut-il ajouter à la procédure SomCar pour avoir un résultat correct ?
3. Remplacer dans ce programme la procédure par une fonction et trouver le résultat fourni par l'algorithme.

Avec une fonction, ce programme devient :

0) DEF FN SomCar (…) : entier
1) X1 ← …
2) X2 ← …
3) … ←X1 + X2
4) Fin SomCar

0) Début **Programme principal**
1) X ← 3
2) Y ← 4
3) … SomCar(X, Y)
3) Ecrire (X,Y, Z)
4) Fin du programme principal

Exercice 9

1. Écrire une procédure qui calcule la somme de deux polynômes de degrés respectifs m et n.
2. Écrire une procédure permettant de calculer du produit de deux polynômes de degrés respectifs m et n.
3. Écrire une procédure permettant de calculer le produit $(P_1 + P_2)*P_3$. où P_1 , P_2 et P_3 sont des polynômes de degrés respectifs m, n et p.

Exercice 10

Écrire un programme appelant une procédure permettant de calculer le point d'intersection de deux droites données par leurs équations
$y = a_1x + b_1$ et $y = a_2x + b_2$.
Avant d'appeler cette procédure il faut bien sûr vérifier que $a_1 \neq a_2$.

CHAPITRE

Chapitre 6

Les traitements avancés

Objectifs :

Utiliser l'analyse modulaire pour résoudre des problèmes de tri et de recherche.

Plan du chapitre :

Leçon 1

Methodes de tri

Objectifs spécifiques :

- Connaître les différentes méthodes de tri.
- Utiliser les différentes méthodes de tri pour résoudre des problèmes.
- Présenter des solutions sous forme d'un algorithme puis d'un programme.

Plan de la leçon :

Leçon 1

Methodes de tri

Une place pour chaque chose et chaque chose à sa place.
Samuel SMILES

I. Introduction

Le problème de tri est un classique de l'informatique ; les méthodes de tri sont utilisées dans différentes applications.
Par exemple :
- classer les élèves par ordre alphabétique ou par ordre de mérite
- mettre en ordre un dictionnaire
- trier l'index d'un livre
- afficher une liste triée d'un correcteur d'orthographe
- …

Dans cette leçon, nous exposerons la problématique du tri de séquences de valeurs et nous nous limiterons à l'étude de quelques méthodes de tri parmi les plus connus, notamment le tri par sélection, le tri à bulles et le tri par insertion.

Pour chaque méthode de tri, nous présentons :
- le principe
- l'analyse du problème, la décomposition en modules, l'élaboration des algorithmes et la traduction en Pascal.

Activité 1

On se donne une suite de N nombres entiers, et on se propose de les trier par ordre croissant. Ainsi, pour N=9, la suite [17,2,9,8,2,10,12,22,8] deviendra [2,2,8,8,9,10,12,17,22]

Définition

Un algorithme de tri est une suite finie d'instructions servant à réordonner une séquence d'éléments suivant un critère fixé à priori. Ce critère est en fait une relation d'ordre total sur les éléments à trier.

La conception d'un algorithme de tri dépend beaucoup plus du support matériel de la séquence de valeurs à trier. Celle-ci peut se trouver en mémoire centrale ou sur une mémoire secondaire.

Ces supports ont des caractéristiques physiques assez différentes :
Les mémoires centrales sont rapides (en nanosecondes) mais d'une taille limitée (en mégaoctets).
Les mémoires secondaires, par contre, sont lentes (en microsecondes) mais de grande taille (en gigaoctets). Par conséquent, les algorithmes de tri vont devoir en tenir compte.

Les algorithmes de tri que nous allons définir traitent des tableaux situés dans la mémoire centrale.

Remarques :

- Il faut faire la distinction entre le tri d'un grand nombre d'éléments et le tri de quelques éléments.

- On peut trier autres données que des entiers. Il suffit de disposer d'un domaine de valeurs muni d'une relation d'ordre totale. On peut donc trier des caractères, des mots en ordre alphabétique...

II. Les méthodes de tri

Dans la suite de ce cours, on choisit de trier les séquences par ordre croissant c'est-à-dire du plus petit au plus grand relativement à un ordre total noté ≤.

II.1 Le tri par sélection

II.1.1 Principe

L'algorithme de tri le plus simple est le *tri par sélection*. Il consiste à trouver l'emplacement de l'élément le plus petit du tableau. Une fois cet emplacement trouvé, on compare son contenu avec T[1] et s'ils sont différents, on permute l'élément de l'emplacement trouvé par l'élément de la première position T[1]. Après ce parcours le premier élément est bien placé.

On recommence le même procédé pour le reste du tableau (T[2],..,T[N]), ainsi on recherche le plus petit élément de cette nouvelle partie du tableau et on l'échange éventuellement avec T[2]. Et ainsi de suite ... jusqu'à la dernière partie du tableau formée par les deux derniers éléments (T[N-1], T[N]).

N.B.

- La recherche du plus petit élément d'un tableau sera assurée par une fonction renvoyant la position du minimum. Ce minimum est éventuellement répété plusieurs fois dans T ; on décidera de choisir le premier.
- L'échange de deux éléments d'un tableau sera assurée par une procédure de permutation.

II.1.2 Résolution du problème : tri par sélection

Soit à trier un tableau T de n entiers initialement dans un ordre quelconque en ordre croissant en utilisant la méthode de tri par sélection.

Exemple :

On considère le tableau T contenant les 10 éléments suivants :

T	17	5	9	8	5	10	12	22	8
	1	2	3	4	5	6	7	8	9

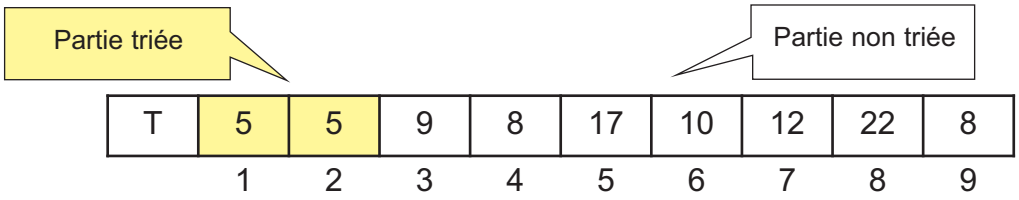
1) - On se pointe à la 1ère case du tableau et on parcourt la totalité de T pour repérer l'indice du minimum c'est-à-dire l'entier 5, ce minimum est répété plusieurs fois dans T, on décidera de choisir le premier c'est-à-dire d'indice 2.

T	17	**5**	9	8	5	10	12	22	8
	1	2	3	4	5	6	7	8	9

- On permute T[1] et ce minimum.
- Après cette opération, on est sûr que T[1] contient la plus petite valeur de T et est donc à sa bonne place.

T	5	17	9	8	5	10	12	22	8
	1	2	3	4	5	6	7	8	9

2) - On cherche la valeur minimale dans (T[2], T[3],…,T[N]) : cette valeur étant 5 et son indice est 5.
- On permute T[2] et ce minimum.



3) - On cherche la valeur minimale dans (T[3],…,T[N]) : cette valeur étant 8, et son indice est 4. Ce minimum est répété plusieurs fois dans T, on décidera de choisir le premier.

T	2	2	9	**8**	17	10	12	22	8
	1	2	3	4	5	6	7	8	9

- On permute T[3] et ce minimum.

T	2	2	8	9	17	10	12	22	8
	1	2	3	4	5	6	7	8	9

4) - On cherche la valeur minimale dans (T[4],…,T[N]) : cette valeur étant 8 et son indice est 9.

T	2	2	8	9	17	10	12	22	**8**
	1	2	3	4	5	6	7	8	9

- On permute T[4] et ce minimum.

T	2	2	8	8	17	10	12	22	**9**
	1	2	3	4	5	6	7	8	9

5) - On cherche la valeur minimale dans (T[5],…,T[N]) : cette valeur étant 9 et son indice est 9.

T	2	2	8	8	17	10	12	22	**9**
	1	2	3	4	5	6	7	8	9

- On permute T[5] et ce minimum.

T	2	2	8	8	9	10	12	22	17
	1	2	3	4	5	6	7	8	9

6) - On cherche la valeur minimale dans (T[6],…,T[N]) : cette valeur étant 10 et son indice est 6.

T	2	2	8	8	9	**10**	12	22	17
	1	2	3	4	5	6	7	8	9

- Puisque la valeur minimale correspond à la première case de la partie non triée du tableau T, l'élément reste dans la même position.

7) - On cherche la valeur minimale dans (T[7],…,T[N]) : cette valeur étant 12 et son indice est 7.

T	2	2	8	8	9	10	12	22	17
	1	2	3	4	5	6	7	8	9

- Puisque la valeur minimale correspond à la première case de la partie non triée du tableau T, l'élément reste dans la même position.

8) - On cherche la valeur minimale dans (T[8],…,T[N]) : cette valeur étant 17 et son indice est 9.

T	2	2	8	8	9	10	12	22	17
	1	2	3	4	5	6	7	8	9

- On permute T[9] et ce minimum.

T	2	2	8	8	9	10	12	17	22
	1	2	3	4	5	6	7	8	9

Résultat : tableau trié

T	2	2	8	8	9	10	12	17	22
	1	2	3	4	5	6	7	8	9

Remarque :

Il faut faire la distinction entre le tri d'un grand nombre d'éléments et le tri de quelques éléments.

Analyse

On choisira à chercher la position du premier minimum quand il y en aura plusieurs occurrences.
Le tableau T sera de type TAB.

	DEF PROC tri_sélection (n : Entier ; VAR T : TAB)	
S	**L.D.E.**	**O.U.**
1	**Résultat** = T_Trié T_Trié = [ ] **Pour** i de 1 à n-1 **Faire** [ppm ←**FN** premposmin (T,n,i)] **Si** (T[ppm] ≠ T[i] **Alors** PROC permut(T[ppm), T[i]) **FinSi** **FinPour**	i ppm premposmin permut
2	**Fin tri_séléction**	

Tableau de déclaration d'un nouveau type

Type
TAB=Tableau de 20 entiers

Tableaux de déclaration des objets de la procédure tri_sélection

Objet	Type/nature	Rôle
i	Entier	Compteur
ppm	Entier	Contiendra la première position du minimum
premposmin	Fonction entière	Renvoie la première position du minimum
permut	Procédure	Permute le contenu de deux variables

	DEF FN premposmin (A: TAB, n, m : Entier ;) : Entier	
S	**L.D.E.**	**O.U.**
2	**Résultat** = (premposmin ← posmin)	i
1	posmin = [posmin ← m] **Pour** j de m+1 à n **Faire**	ppm
	[ ] **Si** A[posmin] > A[j] **Alors**	premposmin
	posmin ← j	permut
	FinSi	
	FinPour	
3	**Fin premposmin**	

Tableaux de déclaration des objets de la fonction premposmin

Objet	Type/nature	Rôle
posmin	Entier	Contient la position du minimum
j	Entier	compteur

	DEF PROC permut (VAR x, y : Entier)	
S	**L.D.E.**	**O.U.**
	Résultat = (x, y)	aux
2	x ← y	x
3	y ← aux	y
1	aux ← x	
4	**Fin permut**	

Tableaux de déclaration des objets de la procédure permut

Objet	Type/nature	Rôle
aux	Entier	Variable auxiliaire nécessaire à la permutation

Algorithmes

```
0)  DEF PROC  tri_sélection (n : Entier ; VAR T : TAB)
1) Pour  i de 1  à  n-1  répéter
        [ppm ← FN premposmin (T, n, i)]  Si  T[ppm] ≠ T[i]  Alors
                                              PROC permut ( T[ppm], T[i] )
                                        FinSi
   FinPour
2) Fin PROC tri_sélection
```

```
0)  DEF FN premposmin (A: TAB, n , m : Entier) : Entier
1) [ posmin ← m]  Pour  j de m+1  à  n  répéter
                      Si A[posmin] > A[j]  Alors
                                          posmin ← j
                      FinSi
                FinPour
2) premposmin  ← posmin
3) Fin premposmin
```

```
0) DEF PROC permut ( VAR x, y : Entier)
1) aux ← x
2)  x  ← y
3)  y  ← aux
4) Fin permut
```

Traduction en Pascal

On suppose que T est un tableau de type TAB.

```
PROCEDURE tri_sélection (n : INTEGER ; VAR T : TAB);
VAR  ppm,i :INTEGER;

PROCEDURE permut(VAR x, y : INTEGER);
VAR  aux : INTEGER;
BEGIN
     aux := x;
     x := y;
     y := aux;
END;

FUNCTION premposmin (A : TAB; n, m : INTEGER):INTEGER;
VAR posmin, j :INTEGER;
BEGIN
     posmin := m;
     FOR j:=m+1 TO n DO
       BEGIN
         IF A[posmin] > A[j] THEN posmin := j;
       END;
     premposmin := posmin ;
END;

BEGIN

FOR i:=1 TO n-1 DO
       BEGIN
         ppm := premposmin (T, n, i) ;
         IF T[ppm]<>T[i] THEN  permut(T[ppm], T[i]);
       END;
END;
```

Remarque :

Il est facile de compter le nombre d'opérations nécessaires pour trier tout le tableau T. A chaque itération, on démarre à l'élément T[i] et on le compare successivement à T[i+1], T[i+2], ..., T[n]. On fait donc n-i comparaisons.
On commence avec i=1et on finit avec i=n-1.
Donc, on fait (n-1) + (n-2) + … + 2 + 1=n(n-1)/2 comparaisons, et au maximum n-1 permutations.

II.2 Le tri à bulles

II.2.1 Principe

Le tri à bulles consiste à faire remonter le plus grand élément du tableau en comparant les éléments successifs. C'est-à-dire qu'on va comparer le 1er et le 2^{ième} élément du tableau, s'ils ne sont pas dans le bon ordre, on les permute, on passe ensuite au 2^{ième} et 3^{ième}, puis au 3^{ième} et 4^{ième} et ainsi de suite jusqu'au (n-1)^{ième} et le n^{ième} éléments.

A la fin du premier parcours, on aura pousser le plus grand élément du tableau vers sa place finale qui est le n^{ième} élément du tableau. On recommence cette opération en parcourant de 1 à n-1 puis de 1 à n-2 et ainsi de suite.

On arrête quand la partie à trier est réduite à un seul élément ou que le tableau est devenu trié et dans ce cas on doit avoir un indicateur qui donne cette information. En effet, l'idée est de vérifier si lors du dernier parcours aucune permutation n'a été faite ce qui signifie que le tableau est devenu trié.

II.2.2 Résolution du problème : tri à bulles

Soit à trier un tableau T de n entiers par ordre croissant en utilisant la méthode de tri à bulles.

Exemple

On considère le tableau T contenant les 5 éléments suivants :

T	3	1	9	0	3
	1	2	3	4	5

1) - On se pointe à la 1ère case du tableau et on compare T[1] et T[2].

T	3	1	9	0	3
	1	2	3	4	5

- Puisque 3 >1, on les permute.

T	1	3	9	0	3
	1	2	3	4	5

2) On compare T[2] et T[3]. Puisque ils sont dans le bon ordre on ne fait rien.

T	1	3	9	0	3
	1	2	3	4	5

3) - On compare T[3] et T[4].

T	1	3	9	0	3
	1	2	3	4	5

- Puisque 9>0, on les permute.

T	1	3	0	9	3
	1	2	3	4	5

4) - On compare T[4] et T[5].

T	1	3	0	9	3
	1	2	3	4	5

- Puisque 9>3, on les permute.

T	0	1	3	3	9
	1	2	3	4	5

On a parcouru les éléments (T[1],T[2], … T[n]) en permutant toute paire d'éléments consécutifs (T[j-1], T[j]) non ordonnés. Ainsi après un parcours, l'élément maximum 9 se retrouve en T[n]. En effet au fur et à mesure des échanges, la méthode fait avancer le plus grand élément rencontré.

A la fin de la première boucle, T[n] contient le plus grand élément du vecteur et cette $n^{ème}$ composante ne doit plus être prise en compte dans la suite du tri puisqu'elle est à sa bonne place.

Il suffit de recommencer ensuite le même processus avec les éléments de T allant de la première composante à n-1, et ainsi de suite on recommence avec les éléments (T[1],T[2], … T[n-2]) jusqu'à épuisement de tous les sous tableaux.

Analyse

	DEF PROC tri_bulles (**n : Entier ; VAR T : TAB**)	
S	**L.D.E.**	**O.U.**
1	**Résultat** = T_Trié Trié = [ ] **Répéter** [Echange ← faux] **Pour** i de 1 à n-1 **faire** [ ] **Si** (T[i]>T[i+1]) **alors** PROC Permut(T[i], T[i+1]) PROC Echange ← vrai **Fin SI** **Fin Pour** **n ← n-1** **Jusqu'à** (n=1) ou non (Echange)	Echange i Permut
2	**Fin tri_bulles**	

Tableau de déclaration d'un nouveau type

Type
TAB=Tableau de 20 entiers

Tableau de déclaration des objets de la procédure tri_bulles

Objet	Type/nature	Rôle
Echange	Bouléen	Reçoit la valeur vrai si une permutation a u lieu
i	Entier	Compteur
T	TAB	Tableau d'entiers
permut	Procédure	Permute le contenu de deux variables

	DEF PROC permut **(VAR x, y : Entier)**	
S	**L.D.E.**	**O.U.**
	Résultat = (x, y)	aux
2	x ← y	x
3	y ← aux	y
1	aux ← x	
4	**Fin permut**	

Tableau de déclaration des objets de la procédure permut

Objet	Type/nature	Rôle
aux	Entier	Variable auxiliaire nécessaire à la permutation

Algorithmes

```
0)  DEF PROC tri_bulles (n : Entier ; VAR T : TAB)
1) Répéter
     Echange ← faux
      Pour i de 1 à n-1 faire
         Si (T[i]>Tt[i+1]) Alors
                              PROC Permut(T[i], T[i+1])
                              PROC Echange ← vrai
         Fin SI
       Fin Pour
     n ← n-1
  Jusqu'à (n=1) ou non (Echange)
2) Fin PROC tri_bulles
```

0) DEF PROC ***permut*** *(VAR x, y : Entier)*
1) aux ← x
2) x ← y
3) y ← aux
4) Fin permut

Traduction en Pascal

Le programme pascal ci-dessous comporte une procédure remplissage et une procédure d'affichage permettant de mettre en œuvre la procédure du tri à bulles.

```
PROGRAM TRI_A_BULLES;
USES WINCRT;
TYPE TAB = ARRAY [1..50] OF INTEGER;
VAR
   T: TAB;
   n: INTEGER;
   i: INTEGER;

PROCEDURE RemplirT(var n:INTEGER ; var T:TAB);
BEGIN
   WRITE('Entrer la taille du tableau : ');
   READLN(n);
   FOR i := 1 TO n DO
    BEGIN
      WRITE('Entrer élément ',i,' : ');
      READLN (T[i]);
    END;
END;

PROCEDURE tri_bulles (n:INTEGER ; var T: TAB);
   VAR
    i: INTEGER;
    Echange: BOOLEAN;

PROCEDURE Permut(var X,Y:INTEGER);
   VAR
    TEMP:INTEGER;
      BEGIN
         TEMP := x;
         x := y;
         y := TEMP;
      END;
```

```
BEGIN
  REPEAT
    Echange:=false;
    FOR i := 1 TO n-1 DO
        BEGIN
          IF T[i] > T[i+1] THEN
                                BEGIN
                                  Permut(T[i],T[i+1]);
                                  Echange:= true;
                                END;
        END;
       n := n-1 ;
     UNTIL (n=1) OR NOT(Echange);
  END;

  PROCEDURE AfficherT(n:INTEGER ; T:TAB);
  BEGIN
    WRITELN('Résultat du programme TRI_A_BULLES :');

    FOR i := 1 TO n DO
     WRITELN('T[', i, ']: ', T[i])
  END;

BEGIN
 RemplirT(n,T);
 TRI_Bulles(n,T);
 AfficherT(n,T);
END.
```

Un cas d'exécution

```
Entrer la taille du tableau : 6
Entrer élément 1 : 14
Entrer élément 2 : 19
Entrer élément 3 : 4
Entrer élément 4 : 7
Entrer élément 5 : 14
Entrer élément 6 : 2
Résultat du programme TRI_A_BULLES :
T[1]: 2
T[2]: 4
T[3]: 7
T[4]: 14
T[5]: 14
T[6]: 19
```

Remarque :

Dans le pire des cas le tri à bulles fait (n-1)+(n-2)+…+2+1 comparaisons et autant de permutations.

II.3 Le tri par insertion

II.3.1 Principe

Le tri par insertion consiste à chercher la position du $i^{ème}$ élément dans la partie du tableau commençant de 1 à i sachant que les i-1 premiers éléments sont triés. Si cette position est i, l'élément est donc à sa bonne place, sinon, supposons que cette position est j. Ce j est forcément entre 1 et i-1. On décale d'un pas vers l'avant (à droite) tous les éléments de j à i-1 puis on insère l'élément d'indice i à la position j.
On commence ce procédé à partir du $2^{ème}$ élément i=2.
Cette méthode ressemble étroitement au rangement d'une main de cartes par un joueur de belotte.

II.3.2 Résolution du problème : tri par insertion

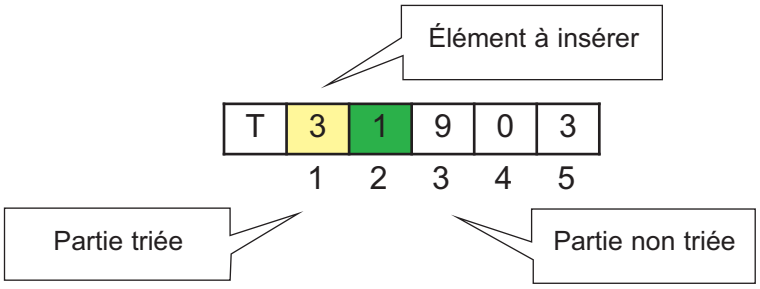
Soit à trier un tableau T de 5 entiers en ordre croissant en utilisant la méthode de tri par insertion.

Pré-analyse
On considère le tableau T contenant les 5 éléments suivants :

T	3	1	9	0	3
	1	2	3	4	5

Au départ, on considère que la partie triée contient le seul premier élément qui vaut 3, le reste formant la partie non triée. Nous commençons donc par le deuxième élément du tableau à savoir l'élément contenant 1 et d'indice 2.

1) On insère l'élément n°2 dans la première partie du tableau à sa place.



- On affecte à la variable auxiliaire TEMP la valeur de l'élément d'indice 2
- On décale d'une case à droite l'élément d'indice 1
- Affecter à la dernière case décalée la valeur de TEMP

T	1	3	9	0	3
	1	2	3	4	5

2) - On insère l'élément n°3 dans la première partie du tableau à sa place.
- Puisque 9>3, donc les trois premiers éléments sont déjà en ordre.

T	1	3	9	0	3
	1	2	3	4	5

3) - On insère l'élément n°4 dans la première partie du tableau à sa place.

T	1	3	9	0	3
	1	2	3	4	5

- On affecte à TEMP la valeur de l'élément d'indice 4.
- Pour insérer l'élément n° 4, nous allons parcourir la partie triée de la droite vers la gauche en décalant d'une case à droite l'élément n° 3, puis l'élément n° 2 et ainsi de suite jusqu'à avoir un élément ayant une valeur inférieure à celle de l'élément à insérer.
- Affecter à la dernière case décalée la valeur de TEMP.

T	0	1	3	9	3
	1	2	3	4	5

4) - On insère l'élément n°5 dans la première partie du tableau à sa bonne place.

T	0	1	3	9	3
	1	2	3	4	5

- On affecte à TEMP la valeur de l'élément d'indice 5.
- On décale d'une case à droite l'élément n°4, jusqu'à avoir un élément inférieur à 3.
- Affecter à la dernière case décalée la valeur de TEMP.

Résultat : tableau trié.

T	0	1	3	3	9
	1	2	3	4	5

Analyse

L'analyse ci-dessous comporte une procédure **Decaler_d** et une procédure **Inserer** permettant de mettre en œuvre la procédure du tri par insertion.

	DEF PROC Tri_insertion (n : Entier ; VAR T : TAB)	
S	**L.D.E.**	**O.U.**
1 2	Résultat = T_trié T_trié = [ ] **Pour** i de 2 **à** n **faire** v ← T[i] j ← i *PROC Decaler_d(T,j,v)* *PROC Inserer(T,j,v)* **Fin Pour** **Fin Tri_insertion**	i v j Decaler_d Inserer

Tableau de déclaration d'un nouveau type

Type
TAB=Tableau de 20 entiers

Tableau de déclaration des objets de la procédure Tri_insertion

Objet	Type/nature	Rôle
i	Entier	Compteur
v	Entier	élément à insérer
j	Entier	Position de l'élément à insérer
Decaler_d	Procédure	Décale les éléments d'un tableau vers la droite
Inserer	Procédure	Insère un élément dans un tableau dans une position bien définie

	DEF PROC Decaler_d (VAR T:TAB; VAR p:Entier ; e:Entier)	
S	**L.D.E.**	**O.U.**
1 2	Résultat = T **T=[ ]Tant que** (T[p-1] > e) **Répéter** T[p] := T[p-1] p:= p-1 **Fin Tantque** **Fin Decaler_d**	T p e

	DEF PROC Inserer (VAR T:TAB; p:Entier ; e:Entier)	
S	**L.D.E.**	**O.U.**
1 2	Résultat = T T[p] := e **Fin Inserer**	T p e

Algorithmes

*0) DEF PROC **Tri_insertion** (n :Entier ; VAR T : TAB)*
1) Pour i de 2 à n faire
v ← T[i]
j ← i
PROC Decaler_d(T,j,v)
PROC Inserer(T,j,v)
Fin Pour
2) Fin PROC Tri_insertion

*0) DEF PROC **Decaler_d** (var T:TAB; var p:Entier ; e:Entier)*
1) Tant que T[p-1] > e Répéter
T[p] := T[p-1]
p:= p-1
Fin tantque
2) Fin PROC Decaler_d

*0) DEF PROC **Inserer** (var T:TAB; p:Entier ; e:Entier)*
1) T[p] := e;
2) Fin PROC Inserer

II.3.3 Traduction en Turbo Pascal

```
PROCEDURE Tri_insertion(n:INTEGER ; var T: TAB);
    VAR  i,j,v: INTEGER;
    PROCEDURE Decaler_d(var T:TAB; var p:INTEGER;e:INTEGER);
      BEGIN
        WHILE T[p-1] > e DO
          BEGIN
            T[p] := T[p-1];
            p:= p-1
          END;
      END;
    PROCEDURE Inserer(var T:TAB; p:INTEGER; e:INTEGER);
      BEGIN
        T[p] := e;
      END;

    BEGIN
      FOR i := 2 TO n DO
       BEGIN
         v := T[i];
         j := i;
         Decaler_d(T,j,v);
         Inserer(T,j,v);
       END;
    END;
```

Remarque :
Dans le pire des cas le tri par insertion fait 1+2+…+n-1 comparaisons et autant de décalages.

En résumé, il fait n(n-1) opérations (comparaisons et décalages confondus) ; on verra plus loin que ce nombre sera considérablement réduit en utilisant la méthode de dichotomie pour la recherche de la position d'insertion.

Retenons

Un algorithme de tri consiste à ranger dans un ordre croissant ou décroissant une suite d'éléments quelconques selon un critère préalablement défini .
On se propose de trier un tableau T par ordre croissant.

La méthode du tri par sélection du tableau T consiste à :

1. Commencer par i=1 et chercher la position pmin du plus petit élément de T. Si T[i] ≠T[pmin] alors on les permute. T[1] est maintenant à sa bonne place.

2. On recommence ces opérations pour la partie du tableau de 2 à n. Après quoi T[2] sera à sa bonne place, et ainsi de suite jusqu'au traitement de la partie formée par les deux derniers éléments. Sachez que si les n-1 éléments du tableau sont bien placés le dernier le sera d'office.

La méthode du tri à bulles du tableau T consiste à :

1. Commencer par i=1 et comparer T[1] et T[2] s'ils ne sont pas dans le bon ordre on les permute puis on passe à la paire T[2] et T[3] puis la paire T[3] et T[4] jusqu'à la dernière paire T[n-1] et T[n]. On aura pousser ainsi le plus grand élément de T vers la dernière position du tableau, donc T[n] est maintenant à sa bonne place donc plus la peine d'y revenir.

2. On recommence les opérations de l'étape 1 en parcourant de 1 à n-2 puis de 1 à n-3 et ainsi de suite. On arrête quand la partie à trier est réduite à un seul élément ou si lors du dernier parcours aucune permutation n'a été faite ; ce qui signifie que le tableau est devenu trié.

La méthode du tri par insertion du tableau T consiste à :

Nous supposons que les éléments de 1 à i-1 sont bien placés (triés), nous allons placer le $i^{ème}$ élément.

1. On cherche sa position d'insertion de l'élément suivant, à savoir celui de la position i, à sa position parmi les i-1 premiers éléments. Si cette position est i, l'élément est donc à sa bonne place, sinon, supposons que cette position est j. On décale d'un pas vers l'avant tous les éléments de j à i-1 puis on insère l'élément d'indice i à la position j.

2. On recommence les opérations de l'étape 1 jusqu'à atteindre la fin du tableau.

Exercices

Exercice 1

On se propose d'écrire une analyse permettant :

1. de saisir les éléments d'un tableau T composé de n chaînes de caractères non vides.
2. de trier le tableau T dans un ordre croissant selon les deux critères suivants :
   - ✓ longueur de la chaîne en premier lieu
   - ✓ ordre alphabétique en cas d'égalité pour les longueurs.

Exercice 2

Il existe plusieurs variantes de l'algorithme du tri à bulles :

Une autre version est le tri bidirectionnel. Elle consiste à parcourir le tableau de gauche à droite, puis de droite à gauche, le changement de direction ayant lieu chaque fois que l'une des extrémités est atteinte. Ainsi, les plus petits éléments du tableau descendent au même rythme que remontent les plus grands éléments.

Question :

Écrire un programme Pascal permettant de saisir n entiers (10<n<30) dans un tableau T et de le trier en utilisant le principe mentionné ci-dessus.

Exercice 3

Nous disposons de deux tableaux Tnom de n chaînes de caractères et Tcouleur de n caractères. Une couleur peut être blanche de caractère 'B' ou noire de caractère 'N'.

Écrire un programme en Pascal qui permet de réarranger les éléments de Tnom et Tcouleur de manière à ce que les éléments de couleur 'B' précèdent les éléments de couleur 'N'. Si deux éléments ont des couleurs identiques , l'ordre alphabétique des chaînes intervient.

Exemple :

(Ali, B)(Salah, N), (Sonia, B), (Tounsi, N), (Salma, N) et (Ahmed, B) sont réarrangés comme suit :
(Ahmed, B), (Ali, B), (Sonia, B), (Salah, N), (Salma, N) et (Tounsi, N).

Exercice 4

Faire une analyse, écrire un algorithme puis la traduction en Pascal d'un programme qui permet de créer un tableau d'entiers croissants à partir de deux tableaux d'entiers non croissants.

Exemple :

V1 = 1, 3, 2, -6
V2 = 0, 4, -5
résultat = -6, -5, 0, 1, 2, 3, 4

Exercice 5

On veut écrire un programme pascal permettant de faire une étude comparative des algorithmes de tri (sélection, bulles et insertion).

Questions :

1. Le programme devra remplir aléatoirement un tableau de N entiers, l'afficher, le trier à l'aide des différentes méthodes de tri proposées (toujours à partir du même tableau initial) et afficher les tableaux triés obtenus.

2. Comparer les temps d'exécution de ces algorithmes pour différentes valeurs de N.

Exercice 6

Écrire, en s'inspirant du tri par sélection, une procédure qui permet de construire à partir d'un tableau T de n entiers un tableau RANG tel que RANG[i] soit l'indice dans T du ieme élément dans l'ordre croissant sans modifier le tableau T.

Exemple :

T	80	50	91	34	20
	1	2	3	4	5

RANG	5	4	2	1	3
	1	2	3	4	5

Leçon 2

Algorithmes de recherche d'un element dans un tableau

Objectifs spécifiques

- Connaître les différentes méthodes de recherches.
- Savoir choisir la méthode de recherche la plus adaptée au problème traité.

Plan de la leçon

Leçon 2

Algorithmes de recherche d'un element dans un tableau

la vraie méthode est la voie par laquelle la vérité elle-même, ou les essences objectives des choses ou leurs idées sont cherchées dans l'ordre dû.

SPINOZA

I. Introduction

On a souvent besoin de rechercher, dans un tableau, un élément donné ou son éventuelle position. Un point particulier à ne pas oublier pour tous les algorithmes est le traitement du cas où l'élément cherché n'est pas dans le tableau. Une autre caractéristique importante d'un algorithme de recherche est son comportement désiré en cas d'éléments identiques (doit-il donner le premier, le dernier ou tous les éléments identiques ?).
Dans cette leçon, nous nous limiterons à l'étude de deux méthodes de recherche parmi les plus connues, notamment la recherche séquentielle et la recherche dichotomique.

II. La recherche séquentielle

Activité 1

Écrire un programme qui saisit un entier naturel n suivi de n entiers à mettre dans un tableau A puis une valeur v, ensuite il appelle une fonction recherche qui cherche si v figure ou non dans le tableau A.

Pré-analyse

Nous disposons d'un tableau A à n éléments. Nous nous proposons de chercher si une valeur v existe dans le tableau A. Nous allons écrire une fonction booléenne qui renvoie VRAI si v existe dans A et FAUX sinon.
La méthode que nous allons adopter consiste à parcourir le tableau A à partir du premier élément et comparer chaque élément de A avec v. Le parcourt s'arrête si on trouve v ou on arrive à la fin du tableau.

Analyse

	Nom = Cherche_elem	
S	**L.D.E.**	**O.U.**
4	**Résultat** = Ecrire ("Existe = ", exist)	exist
3	exist ← FN Recherche (A, n, v)	A
2	v = Donnée	n
1	A = [n = Donnée] **Pour** i de 1 à n **Faire**	v
	A[i] = Donnée	i
	FinPour	Recherche
5	**Fin cherche_elem**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
exist	Booléen	Paramètre effectif résultat booléen de la recherche
A	Tableau d'entiers	Tableau des éléments
n	Entier	Nombre d'éléments de A
v	Entier	Élément à rechercher
i	Entier	Compteur
Recherche	Fonction	Renvoie VRAI si v existe dans A et FAUX sinon.

Algorithme

1) Début Cherche_elem
2)Lire (n)
3)Pour i de 1 à n Répéter
Lire (A(i))
FinPour
4)Lire (v)
5)exist ← FN Recherche (A, n, v)
6)Ecrire ("Existe = ", exist)
7)Fin Chercher_elem.

Analyse de la fonction Recherche

	DEF FN Recherche (n, p : Entier ; TE : tableau de 20 entiers) : Booléen	
S	**L.D.E.**	**O.U.**
	Résultat = Rechercher	TE
2	Recherche = [ ]**Si** (TE[i] = p) **Alors**	n
	Recherche ← VRAI	p
	Sinon	i
	Recherche ← FAUX	
	FinSi	
1	i = [i← 0] **Répéter**	
	i ← i +1	
	Jusqu'à (TE[i] = p) **OU** (i = n)	
3	**Fin Recherche**	

Tableau de déclaration des objets locaux

Objet	Type/nature	Rôle
i	Entier	Compteur

Algorithme

0) DEF FN Recherche (n, p : entier ; TE : tableau de 20 entiers) : booléen
1) [i ← 0] Répéter
i ← i +1
Jusqu'à (TE[i] = p) OU (i = n)
2) Recherche = Si TE[i] = p Alors
Recherche ← VRAI
Sinon
Recherche ← FAUX
FinSi
3) Fin Recherche

Traduction en Pascal

```
PROGRAM Cherche_elem;
USES WINCRT;
TYPE Typtab=ARRAY [1..20] OF INTEGER ;
VARexist : BOOLEAN ;
   Indice : INTEGER;
   T : Typtab ;
   n,i,v : INTEGER;

FUNCTION Recherche (n,p:INTEGER ; TE:Typtab):BOOLEAN;
VAR
  i : INTEGER ;
BEGIN
  Exist := False ;
  i := 0 ;
  REPEAT
     i := i + 1 ;
  UNTIL  (TE[i] = p)OR( i = n );
  IF  TE[i] = p THEN
                  Recherche:= TRUE
               ELSE
                  Recherche:= FALSE ;
END ;
```

```
BEGIN
WRITE('Entrer la taille du tableau : ');
READLN(n);
  FOR i := 1 TO n DO
    BEGIN
     WRITE('Entrer un élément :  ');
     READLN(T[i]);
   END;
WRITE('Entrer l''élément à rechercher :  ');
READLN(v);
  Exist:=Recherche (n, v, T);
  WRITELN ('Exist =', exist);
END.
```

Un cas d'exécution

```
Entrer la taille du tableau : 7
Entrer un élément :  3
Entrer un élément :  8
Entrer un élément :  9
Entrer un élément :  6
Entrer un élément :  7
Entrer un élément :  6
Entrer un élément :  9
Entrer l'élément à rechercher :  6
Exist =TRUE
```

Remarque :

Cet algorithme effectue au maximum n comparaisons.

II.1 Définition

La méthode de recherche séquentielle d'un élément dans un tableau consiste à parcourir le tableau élément par élément en les comparant avec l'élément à chercher jusqu'à trouver ce dernier ou achever le tableau.

II.2 Application

Développez un programme qui saisit un entier naturel n suivi de n réels à mettre dans un tableau A, puis une valeur v dont on veut chercher la dernière position dans le tableau si elle y figure. Dans le cas où v ne figure pas dans le tableau le programme affichera la position 0.

Pré-analyse

Nous disposons d'un tableau A à n éléments. Nous nous proposons de chercher la dernière position d'une valeur v si elle existe dans le tableau A. Nous allons écrire une fonction booléenne qui renvoie la dernière position si v existe dans A et la valeur 0 sinon.
La méthode que nous allons adopter consiste à parcourir le tableau A à partir du dernier élément et comparer chaque élément de A avec v. Le parcourt s'arrête après avoir trouvé v ou arrivé au début du tableau.

Analyse

	Nom = Cherche_der_pos_elem	
S	**L.D.E.**	**O.U.**
4	**Résultat** = Ecrire ("Dernière Position = ", pos)	pos
3	pos ← FN Recherche_der_pos (A, n, v)	A
2	v = Donnée	n
1	(A,n) = [n = Donnée] **Pour** i de 1 à n **Répéter**	v
	T[i] = Donnée	i
	FinPour	Recherche_der_pos
5	**Fin cherche_der_pos_elem**	

Tableau de déclaration des objets

Objet	Type/nature	Rôle
Pos	Entier	Dernière position de l'élément v à chercher
A	Tableau d'entiers	Tableau des éléments
n	Entier	Nombre d'éléments de A
v	Entier	Élément à rechercher
i	Entier	Compteur
Recherche_der_pos	fonction	Renvoie VRAI si v existe dans A et FAUX sinon

Algorithme

0) Début Cherche_der_pos_elem
1) Lire (n)
Pour i de 1 à n Répéter
Lire (A(i))
FinPour
2) Lire (v)
3) pos ← FN Recherche_der_pos (A, n, v)
4) Ecrire ("Dernière Position = ", pos)
5) Fin Chercher_der_pos_elem

Analyse de la fonction Recherche_der_pos

	DEF FN Recherche_der_pos (TE : tableau de 20 entiers, n,m : Entier) : Entier	
S	**L.D.E.**	**O.U.**
4	**Résultat** = Recherche_der_pos	TE
3	Recherche_der_pos ← p	n
2	p = [p← 0] **Si** (TE[i] = m) **Alors**	m
	p ← i	i
	FinSi	p
1	[i ← n+1] **Répéter**	
	i ← i -1	
	Jusqu'à (TE[i] = m) OU (i = 1)	
5	**Fin Recherche_der_pos**	

Tableau de déclaration des objets locaux

Objet	Type/nature	Rôle
p	Entier	Dernière position
i	Entier	Compteur

Algorithme

0) DEF FN Recherchder_der_pos (TE : tableau de 20 entiers, n, m : entier) : entier
1) [i ← n+1] Répéter
i ← i -1
Jusqu'à (TE[i] = m) OU (i = 1)

2) p = [p ← 0] Si TE[i] = m
Alors
p ← i
FinSi
3) Recherchder_der_pos←P
4) Fin Recherche-der_pos

Traduction en Pascal

```
PROGRAM Cherche_der_pos_elem;
USES WINCRT;
TYPE Typtab=ARRAY [1..20] OF INTEGER ;
VAR
    T : Typtab ;
    pos,n,v,i: INTEGER;
```

```
PROGRAM Cherche_der_pos_elem;
USES WINCRT;
TYPE Typtab=ARRAY [1..20] OF INTEGER ;
VAR
    T : Typtab ;
    pos,n,v,i: INTEGER;

FUNCTION Recherche_der_pos (TE:Typtab; n,m:INTEGER):INTEGER;
VAR
  p,i : INTEGER ;
BEGIN
  i := n+1;
  REPEAT
     i := i - 1;
  UNTIL(TE[i] = m)OR(i = 1);
  p:=0;
  IF  TE[i] = m THEN
                    p:= i;
  Recherche_der_pos:=P;
END ;

BEGIN
  WRITE('Entrer la taille du tableau : '); READLN(n);
  FOR i := 1 TO n DO
    BEGIN
      WRITE('Entrer un élément : ');
      READLN(T[i]);
    END;
WRITE('Entrer l''élément à rechercher : ');READLN(v);
  pos:=Recherche_der_pos(T, n, v);
  WRITELN ('Dernière Position =', pos);
END.
```

Un cas d'exécution

```
Entrer la taille du tableau : 6
Entrer un élément : 4
Entrer un élément : 8
Entrer un élément : 9
Entrer un élément : 7
Entrer un élément : 3
Entrer un élément : 8
Entrer l'élément à rechercher : 8
Dernière Position =6
```

III. La recherche dichotomique

Activité 1

On vous propose de chercher dans un dictionnaire la signification du mot informatique. Décrivez en détail comment vous allez faire cette recherche.

La première action est d'ouvrir le dictionnaire "en son milieu" et de voir si on'est pas tombé par hasard sur la page comportant le mot informatique.

Dans le cas contraire, en comparant la lettre initiale du mot informatique avec celle affichée en haut de la page du dictionnaire on saura si on doit rechercher ce mot dans la partie gauche ou la partie droite.

On continue ce processus jusqu'à ouvrir la page contenant le mot informatique. Cette approche profite du fait qu'un dictionnaire est trié.

Activité 2

Effectuer une analyse, un algorithme et la traduction en Pascal de la fonction intitulée RECHERCHE qui utilise la méthode de recherche dichotomique pour vérifier l'existence d'un élément donné dans un tableau de n entiers triés par ordre croissant.

Pré-analyse
On regarde l'élément au milieu du tableau.
- s'il est égal à la valeur cherchée, l'élément cherché est existant.
- s'il est inférieur à la valeur recherchée, il ne reste à traiter que la moitié droite du tableau.
- s'il est supérieur à la valeur recherchée, il ne reste à traiter que la moitié gauche du tableau.

On continue ainsi la recherche en diminuant à chaque fois de moitié le nombre d'éléments du tableau restants à traiter.

DEF FN Recherche (n, m : Entier ; TE:Typtab): Bouléen

S	L.D.E.	O.U.
1	**Résultat** = Result_recherche Result_recherche = [a←1, b←n, Result_recherche←Faux] **Répéter** [p←(a+b) DIV 2] **Si** (m=TE[p]) **Alors** Result_recherche←Vrai **Sinon Si** (m < TE[p]) **Alors** b ←p-1 **Sinon** a←p+1 **FinSi** **Jusqu'à** (Result_recherche) OU (a>b)	TE n m a b p
2	**Fin Recherche**	

Tableau de déclaration des objets locaux

Objet	Type/nature	Rôle
a	Entier	Borne droite
b	Entier	Borne gauche
p	Entier	compteur

Algorithme

```
0)DEF FN Recherche (n,m:Entier ; TE:Typtab): Bouléen
1)a←1
  b←n
  Result_recherche←Faux
  Répéter
        p←(a+b) DIV 2
        Si (m=TE[p]) Alors  Result_recherche←Vrai
        Sinon Si (m < TE[p]) Alors
                                  b ←p-1
        Sinon
            a←p+1
        FinSi
    Jusqu'à (Result_recherche) OU (a>b)
 2)Fin Recherche
```

Traduction en Pascal

```
PROGRAM Cherche_elem;
USES WINCRT;
TYPE Typtab=ARRAY [1..20] OF INTEGER;
VAR  Ok: BOOLEAN;
     T : Typtab;
     n,i,p : INTEGER;

FUNCTION Recherche (n,m:INTEGER ; TE:Typtab): BOOLEAN;
VAR
   a,b :INTEGER;
Result_recherche:BOOLEAN ;
BEGIN
  a:=1;
  b:=n;
  Result_recherche:=False ;
```

```
  REPEAT
    p := (a+b) DIV 2 ;
    IF m = TE[p] THEN Result_recherche:=True
    ELSE IF m < TE[p] THEN
                        b :=p-1
    ELSE
      a:=p+1;
   UNTIL (Result_recherche) OR (a>b);
   Recherche:=Result_recherche;
END;

BEGIN
WRITE('Entrer la taille du tableau : ');
READLN (n);
FOR   i := 1 TO n DO
   BEGIN
       WRITE('Entrer un élément : ');
       READLN (T[i]);
   END;
WRITE('Entrer l''élément à rechercher : ');
READLN (p);
OK:= Recherche (n,p,T);
WRITELN ('Exist = ' ,OK);
END.
```

Un cas d'exécution

```
Entrer la taille du tableau : 7
Entrer un élément : 12
Entrer un élément : 15
Entrer un élément : 15
Entrer un élément : 19
Entrer un élément : 25
Entrer un élément : 27
Entrer un élément : 99
Entrer l'élément à rechercher : 25
Exist = TRUE
```

Remarques :

- Nous pouvons constater qu'à la première étape, nous devons prendre le milieu d'un tableau de n éléments, à l'étape suivante le milieu d'un tableau de n/2 éléments, puis de n/4 éléments, ...
- La recherche s'arrête quand le nombre d'éléments est réduit à 1 ou l'élément est trouvé.

Définition

En algorithmique, la dichotomie (du grec « couper en deux ») est un processus itératif ou récursif de recherche où à chaque étape l'espace de recherche est restreint à l'une de deux parties.
On suppose bien sûr qu'il existe un test relativement simple permettant à chaque étape de déterminer l'une des deux parties dans laquelle peut se trouver l'élément.

L'algorithme s'applique typiquement à la recherche d'un élément dans un ensemble fini ordonné.

La dichotomie peut être vue comme une variante simplifiée de la stratégie plus générale diviser pour régner (en anglais, divide and conquer).

Retenons

– La méthode de la recherche séquentielle consiste à parcourir le tableau progressivement du début vers la fin en les comparant avec l'élément à chercher jusqu'à trouver ce dernier ou achever le tableau.

L'algorithme de recherche séquentielle n'utilise comme information que le test d'égalité sur les éléments, avec deux résultats possibles : égalité ou non égalité.

– La méthode de la recherche dichotomique consiste à chercher un élément dans le cas d'un tableau trié.

On compare l'élément à chercher à l'élément central du tableau, s'ils sont différents, un test permet de trouver dans quelle moitié du tableau on peut trouver l'élément. On continue ce processus jusqu'à trouver l'élément ou bien on arrive à un sous-tableau de taille 1.

Exercices

Exercice 1

Effectuer une analyse, un algorithme et la traduction en Pascal du programme intitulé ANNUAIRE1 qui, saisit un nom puis détermine le numéro de téléphone correspondant.

Avec deux tableaux Tnom et Ttel, indicés en parallèle, le numéro de téléphone de Tnom[i] étant Ttel[i], on peut faire un annuaire téléphonique. L'information à rechercher est le numéro de téléphone.

Exemple :

Tnom	Index	Ttel
Jamel	1	71222
Naoufel	2	75123
Imed	3	73632
Ghazi	4	72098
Salma	5	78453
Ibtissem	6	72331
Hakim	7	72776
Afef	8	77345
Abdelkader	9	78080
Fadhila	10	77777
Zoubeir	11	72987

Si le nom est Salma alors le numéro de téléphone correspondant est **78453**.

Exercice 2

Supposons que la table des noms est triée par ordre alphabétique (comme l'annuaire des PTT). Au lieu de faire une recherche séquentiellement, utiliser la méthode de recherche dichotomique pour résoudre l'exercice 1.

Exercice 3

Écrire un programme qui lit n caractères introduits au clavier par l'utilisateur et les place au fur et à mesure dans un tableau T. Ensuite le programme recherche dans le tableau la plus longue suite de caractères identiques et affiche le caractère concerné ainsi que le nombre de fois qu'il est répété. On suppose que les caractères tapés par l'utilisateur sont différents des blancs.

Exemple :

si on introduit (a a b c c e d e e e e e f f g a a a),
le programme doit afficher (e,5).

Exercice 4

Écrire un programme en Pascal qui réalise le traitement suivant :
Compactage d'un tableau d'entiers positifs ou nuls : remplacer chaque suite de zéros par le nombre de zéros multiplié par -1.

Exemple :
Vecteur = 1 2 3 0 0 0 2 0 0
résultats = 1 2 3 -3 2 -2

Exercice 5

On veut insérer un nouveau élément dans un tableau trié de n entiers en maintenant le tableau ordonné. Pour insérer un nouvel élément dans le tableau, il faut d'abord trouver son emplacement par une recherche dichotomique, puis décaler tous les éléments pour pouvoir insérer le nouveau élément au bon endroit.
Écrire un programme en Pascal, qui permet d'insérer un élément dans un tableau trié.

Exercice 6

Écrire un programme Pascal permettant :
- d'insérer un réel x dans un tableau à la position p. Si le tableau est plein le programme doit d'abord construire un nouveau tableau dont le nombre d'éléments est 10% supérieur au tableau initial.
- de supprimer l'élément à la position p et de décaler le reste des éléments T[p+1..n] vers la gauche.

BIBLIOGRAPHIE

[BOR] Borland International Inc.
Turbo Pascal 6.0 (Guide du programmeur)
[BOR] Borland International Inc.
Turbo Pascal 6.0 (Guide de référence)
[BOY] J.BOYER, J.-D. LAMOITIER, M. TREILLET.
Systèmes PC.DOS et MS.DOS version 2 à 4
Editests. 1987
[CAR] R. CARRE, J.-F. DEGREMONT, M. GROSS, J.-M. PIERREL, G. SABAH
Langage humain et machine
Presses CNRS 1990
[CHA] G. CHATY, J.VICARD
L'algorithmique de la pratique à la théorie.
CEDIC. 1983
[ENG] A. ENGEL
Mathématique élémentaire d'un point de vue algorithmique.
CEDIC. 1981
[ESP] B. D'ESPAGNAT
Penser la science ou les enjeux du savoir
DUNOD 1990
[KNU] D. KNUTH
The Art Of Computer Programming Vol.1, 2 et3.
ADDISON WESLEY . 1968
Vol. 3 : Sorting and Searching
[LAU] J.-P. LAURENT
Initiation à l'analyse et à la programmation.
DUNOD. 1985
[LIG] P. LIGNELET
Algorithmique. Tome 1 et Tome 2.
MASSON. 1981
[MEY] J.-J. MEYER
Pratique du Turbo Pascal (Créez vos progiciels).
Éditions Radio. 1987
[MEY] J.-J. MEYER
Pratique du Turbo Pascal (Créez vos progiciels).
Éditions Radio. 1987
[PAI] C. PAIR
Sciences et Pratiques de l'Informatique.
BORDAS. 1987
[ROY] D. ROY
Routines en Turbo Pascal.
Éditions P.S.I. 1989
[TRU] J-RUSS. Dictionnaire de philosophie
Bordas 2004
[TRU] J-P. TRUMBLAY, R. B. BUNT, P. G. SORENSON
Logique de programmation
Traduction de G. DUPUIS et L. VAN BUSKIRK
McGraw-Hill. 1985
[WIR] N. WIRTH
Introduction à la programmation systématique.
MASSON. 1986

RÉFÉRENCES WEB

- http://fr.wikipedia.org
- http://www.commentcamarche.net

Annexe

Table des codes ASCII.

HEX	DEC	CAR	HEX	DEC	CAR	HEX	DEC	CAR	HEX	DEC	CAR
0	0	NUL	20	32		40	64	@	60	96	`
1	1	SQH	21	33	!	41	65	A	61	97	a
2	2	STX	22	34	"	42	66	B	62	98	b
3	3	ETX	23	35	#	43	67	C	63	99	c
4	4	EOT	24	36	$	44	68	D	64	100	d
5	5	ENQ	25	37	%	45	69	E	65	101	e
6	6	ACK	26	38	&	46	70	F	66	102	f
7	7	BEL	27	39	'	47	71	G	67	103	g
8	8	BS	28	40	(	48	72	H	68	104	h
9	9	HT	29	41	)	49	73	I	69	405	i
A	10	LF	2A	42	*	4A	74	J	6A	106	j
B	11	VT	2B	43	+	4B	75	K	6B	107	k
C	12	FF	2C	44	,	4C	76	L	6C	108	l
D	13	CR	2D	45	-	4D	77	M	6D	109	m
E	14	SO	2E	46	.	4E	78	N	6E	110	n
F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	\	7C	124	|
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94	^	7E	126	~
1F	31	US	3F	63	?	5F	95	_	7F	127	□

Les autres pages de codes ne sont pas standards, nous ne les reproduisons pas.